NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROPROCESSOR-BASED, SOLAR CELL PARAMETER MEASUREMENT SYSTEM

by

Robert R. Oxborrow

June 1988

Thesis Advisor Sherif Michael

Unclassified

ecurity classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report	
2b Declassification Downgrading Schedule		Approved for public release: distribution is unlimited.	
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(if applicable)* 39	7a Name of Monitoring Organization Naval Postgraduate School	
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	
8a Name of Funding Sponsoring Organization	8b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number	
8c Address *(city, state, and ZIP code)*		10 Source of Funding Numbers	
		Program Element No	Project No / Task No / Work Unit Accession No

11 Title *(include security classification)* A MICROPROCESSOR-BASED, SOLAR CELL PARAMETER MEASUREMENT SYSTEM

12 Personal Author(s) Robert R. Oxborrow

13a Type of Report	13b Time Covered	14 Date of Report *(year, month, day)*	15 Page Count
Master's Thesis	From To	June 1988	90

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Solar Cells, Space Measurement, I-V Curves, Autonomous Control

19 Abstract *(continue on reverse if necessary and identify by block number)*

The effects of the space environment on solar cells has, to date, been largely modeled and approximated in the design of solar arrays. Restrictions such as weight and cost have precluded direct analysis of the long term effects of radiation in space. At the Naval Postgraduate School (NPS), a simple circuit has been devised which facilitates in situ data collection and analysis of these effects. The circuit includes an op-amp and a high beta transistor for cell voltage biasing. When coupled to a microprocessor-based controller system, this circuit has the capability to measure and store data pertaining to solar cell performance I-V curves. The complete system consists of an NSC 800 microprocessor, D A and A D components, analog multiplexers and demultiplexers, biasing transistors and op-amps. This design provides a compact, low power, accurate method for I-V measurement and data storage. Such a system may be used to observe and monitor an array of test cells and their performance degradation in both the space environment and terrestrial applications.

20 Distribution Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual Sherif Michael	22b Telephone *(include Area code)* (408) 646-2252	22c Office Symbol 62Mi

D FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

A Microprocessor-Based, Solar Cell
Parameter Measurement System

by

Robert R. Oxborrow
Lieutenant, United States Navy
B.S., United States Naval Academy, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE, ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
1988, June 06

ABSTRACT

The effects of the space environment on solar cells has, to date, been largely modeled and approximated in the design of solar arrays. Restrictions such as weight and cost have precluded direct analysis of the long term effects of radiation in space. At the Naval Postgraduate School (NPS), a simple circuit has been devised which facilitates in situ data collection and analysis of these effects. The circuit includes an op-amp and a high beta transistor for cell voltage biasing. When coupled to a microprocessor-based controller system, this circuit has the capability to measure and store data pertaining to solar cell performance I-V curves. The complete system consists of an NSC 800 microprocessor, D/A and A/D components, analog multiplexers and demultiplexers, biasing transistors and op-amps. This design provides a compact, low power, accurate method for I-V measurement and data storage. Such a system may be used to observe and monitor an array of test cells and their performance/degradation in both the space environment and terrestrial applications.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

There are a number of people who have listened, offered advice, and provided assistance in the completion of this project. However, there are a few of individuals without whom this work would not have reached its present form.

First, I wish to thank Dr. Michael for his calm demeanor, despite my level of anxiety, his answers to many questions, his contribution to my understanding and learning, and, of course, for the research money to build the project.

I thank Dave Rigmaiden for providing me with some of his energy, time and great skills in microprocessor operation and application, and in the use of associated TTL and CMOS hardware. Dave built and tested the version of controller used for the project.

I also thank Charlie Cameron for his patience, time, and vast knowledge of the 'C' language, which he freely provided for my benefit in developing the solar cell array test circuit. In addition, Charlie provided the microprocessor controller standup and operating routines which were modified for use in this project.

Last, but not least, I wish to thank my wife, who understands, and always supports my endeavors.

I. INTRODUCTION

A. THE UBIQUITOUS SOLAR CELL

The photovoltaic effect, upon which solar cells depend for their operation, was first reported by Becquerel, in 1839. He observed a light-dependent voltage between two electrodes immersed in an electrolyte. The effect was observed in the solid, selenium, in 1876. Photocells made of selenium and cuprous oxide were soon developed [Ref. 1: p.2]. Bell Telephone Laboratories began theoretical research on the photovoltaic effect in the 1930's. During the 1940's experiments with silicon accelerated development of electrical devices utilizing semiconductors. In 1954 the first practical solar cell was produced. The major stumbling block in development of this cell was the production of pure silicon crystal material. Breakthroughs by Czochralski in pure crystal growing and by Fuller and Ditzenberger in high-temperature vapor diffusion to form p-n junctions brought forth the necessary technology for successful semiconductor devices [Ref. 2: p.1.2-1]. The first cells were approximately 3 cm diameter circular wafers, resulting from the maximum diameter crystal that could be grown with existing technology. Conversion efficiency was on the order of six to ten percent.

While the solar cell was first considered only for terrestrial applications, the advantages of light weight, small size, and planar design destined this device to play a major role in the operation of spacecraft, and indeed, this application was by far the major use of the solar cell for over ten years.

Vanguard I, launched on March 17, 1958, became the first solar powered earth satellite. The array consisted of six solar panels distributed around the satellite body, each made of 18 p-n 2.0 x 0.5 cm cells. The system provided less than one watt of power, and operated for over six years in orbit [Ref. 2: p.1.1-1]. Since this austere beginning, solar cell arrays have been a major source of power for a multitude of spacecraft and provided them with from less than a watt to tens of kilowatts of operating power. As the power requirements and complexity of spacecraft have increased, the development of solar cell technology has kept pace. New materials, dopants, surface preparations, and hardware have been developed. Understanding of the hazards of radiation from such sources as the sun, Van Allen Belts, and deep space has prompted the introduction of new adhesives, substrates, and coverglass materials.

Throughout the 1960's emphasis was placed on increasing radiation resistance and decreasing array weight and cost. For almost ten years little progress was made in the development of more efficient solar cells [Ref. 2: p.1.2-1]. In the early 1970's new compounds such as Gallium-Arsenide (GaAs), an optimized contact gridline system, front surface texturing, and new anti-reflective coatings, such as tantalum pentoxide (Ta_2O_5), introduced new "high efficiency" cells with conversion efficiencies of up to sixteen percent [Ref. 2: p.1.2-2]. These developments, coupled with the search for new and better energy sources, reawakened the interest in terrestrial applications for the solar cell [Ref. 1: p.2]. A major concern in the development of these new cells and associated hardware has been the testing and analysis of these devices' performance after prolonged exposure to the space environment, and, to a lesser extent, the earth environment.

B. SOLAR CELL POWER

Solar cells are essentially large p-n diodes, and, as such, possess performance characteristics that are most readily expressed in three parameters. These three parameters are short-circuit current (I_{sc}), open-circuit voltage (V_{oc}), and fill factor (FF). In the ideal case, I_{sc} would equal I_L, the light-generated current. V_{oc} may be defined by:

$$V_{oc} = \frac{kT}{q} \ln\left(\frac{I_L}{I_O} + 1 \right) \tag{1.1}$$

where k = *Boltzmann's Constant*, q = the charge of an electron, T = absolute temperature, and I_o represents the saturation current [Ref. 1: p.79]. The dependence of V_{oc} on I_o makes this voltage parameter also dependent upon the properties of the semiconductor from which the cell is manufactured. I_o may vary with time for a given material; the result of exposure to radiation, age, heat, etc.. Likewise, I_L may vary with light intensity. Fluctuation of these parameters produces varying voltage values which lie along a characteristic I-V curve. As current through the diode, or cell, decreases from I_{sc}, voltage begins to increase, rapidly, at first, until I_L approaches I_O. As this occurs, voltage across the p-n junction rapidly stabilizes at V_{oc}, as may be seen in Figure 1 on page 3. This effect produces the characteristic knee on an I-V curve. The operating point which maximizes the output power of the cell ($V_{mp.}$, $I_{mp.}$) is found on this knee.

FF, a measure of how "square" the output characteristics of the diode, or cell are, is defined by

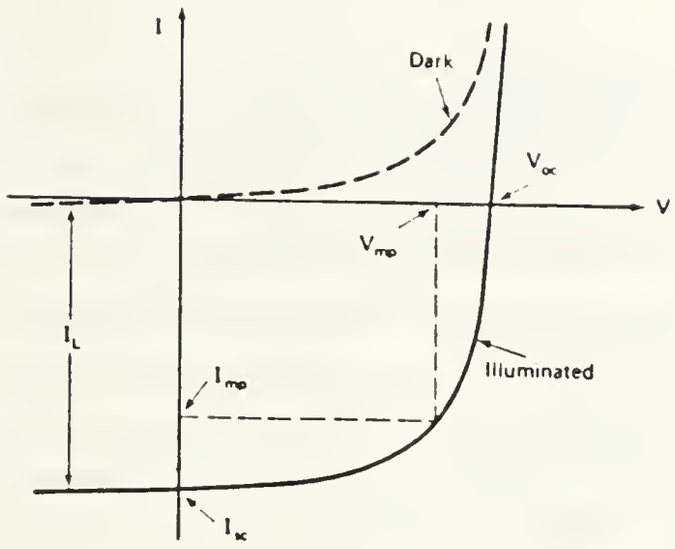


Figure 1. Typical p-n junction diode I-V curve. Ref. 1: p.79

$$FF = \frac{V_{mp} I_{mp.}}{V_{oc} I_{sc}} \tag{1.2}$$

[Ref. 1: p.80]. Optimally, FF is a function of only V_{oc} since I_{sc} is fixed for a given device. The energy conversion efficiency of a solar cell, then, is

$$\eta = \frac{V_{mp.} I_{mp.}}{Pin} = \frac{V_{oc} I_{sc} FF}{Pin} \tag{1.3}$$

where P_{in} is the total incident light power on the cell [Ref. 1: p.81]. Maximum η occurs at the maximum power point ($P_{mp.}$). Common commercial cell efficiencies are in the range of 12 to 16 percent.

C. SOLAR CELL CALIBRATION

The calibration of solar cells to produce "standard" cells is necessary for two reasons. First, to determine the absolute value of the solar constant over the spectral response region of solar cells, and second, to accurately establish the light intensity of solar simulators. Initial solar cell measurements were made outside, on a sunny day, with volt-ohmmeters, pencil, and paper. These "fair weather" tests were soon found inadequate for the accuracy desired in analysis and deficient in their consideration of the effects of the atmosphere on solar radiation.

Solar cells and array assemblies designed for spacecraft were tested under laboratory conditions, illuminated by incandescent tungsten lamps. However, it was found that the color temperature of these lamps, 2700-3400K, were much cooler than the sun, about

6000K, at air-mass-zero (AM0). Further, the spectral composition of the sun was markedly different from that of the tungsten lamps, which contained large infrared components. It was thought that water filters would aid in alleviating some of these spectral problems, but these created even more problems and were abandoned. The stability and reliability of tungsten lamps outweighed the spectral shortcomings of the device through the 1960's. Calibration of these lamps required closely controlled filament voltage to control color temperature, and intensity adjustment by comparison with specially calibrated solar cells. These cells were measured in natural sunlight with a pyrheliometer, a thermopile designed specifically for measuring solar flux. Lamps calibrated with this scheme produced cells which were tested under "Standard Tungsten Test Conditions" (unfiltered tungsten light of 2800K±50K, equivalent to 100 mW/cm² solar radiation at 28°C cell temperature). [Ref. 2: p.11.2-1]

Standard Tungsten Test Conditions were based upon the effect of natural sunlight on solar cells under normal, but arbitrary outdoor conditions. "Standard" solar cells were measured under light at any intensity, and the results extrapolated to 100 mW/cm². Natural sunlight intensity was measured with standard meteorological equipment, which suffered from some accuracy limitations. Cells were measured in collimated sunlight, to eliminate the effects of sky background, or corrected by application of a correction factor based upon the ratio of short circuit currents of a cell measured in uncollimated light to those measured in collimated light. Such calibrated cells were used as "standard" calibration devices for laboratory tungsten illuminators. [Ref. 2: p.11.2-1]

A number of problems and inaccuracies were readily apparent under this system. Natural sunlight conditions at test sites varied in both intensity and spectral content so correlation from one day to the next was poor. The correlation between test sites was worse. Standard cell calibration was then performed at the Smithsonian Institute Solar Observatory, near Los Angeles, California, where data on sunlight conditions and spectra had been collected for twenty-five years. The altitude of the site is 7516 ft, adjacent to the Mohave Desert and characterized by relatively clear skies and low humidities. After the improvement of outdoor illumination condition standards, the problems of color temperature and tungsten sources were addressed through the use of color temperature meters which were used for monitoring light and color temperature adjustments in tungsten lamp voltages. [Ref. 2: p.11.2-1]

It was believed that this calibration methodology, more reproducible and accurate than previous schemes, was sufficient to achieve adequate extrapolation of results to

AM0 conditions. However, in 1961 it was discovered that efforts to improve solar cell efficiencies had significantly shifted spectral response toward the red. Cells and panels measured under sources calibrated against standard cells were resulting in errors of 15 to 20 percent due to the different spectral responses between standards and new cells. Government and industry began a test method standardization program which soon solved some problem areas and defined others. New standard cells were developed and the AIEE established a committee which prepared specifications for measurement of solar cells using simulated solar radiation conditions. [Ref. 2: p.11.2-2]

Attempts were made to achieve the greatest possible accuracy in solar simulation and standards throughout the 1960's. High-altitude balloon flights seemed to have the highest accuracy and became the definitive light intensity standard. The development of solar simulators also progressed rapidly. However, the unavailability of space-calibrated cells to verify simulator performance degraded confidence in the accuracy of these machines. The most widely used solar simulators for cell and array testing since the late 1960's have been the X-25 series solar simulators developed by the Spectrolab Division of Textron Electronics, Inc.. These simulators, and those developed since, use high-power, high-pressure Xenon arc lamps. Smaller, continuously operating lamps uniformly illuminate an area up to nearly 0.07 m^2, illuminating single cells, while Large Area Pulsed Solar Simulators (LAPSS), are used to test arrays up to 5 m^2, permitting a few milliseconds of illumination by radiation closely matched to AM0 conditions. [Ref. 2: p.11.2-1]

D. HIGH-ALTITUDE BALLOON CALIBRATION

Solar cells do not utilize all the energy available in the conversion of light energy to electricity. Various elements of the solar spectrum are absorbed and reflected by the specialized materials from which solar cells are made. Great effort has been expended to produce solar simulators which simulate the intensity of the sun as well as its spectrum. Errors in either could result in an overweight array design for a given application, or a system which would prematurely degrade and become power deficient.

The Jet Propulsion Laboratory (JPL) has been producing calibrated reference solar cells through its solar cell calibration program for over twenty-four years. This program produces reference standard cells, with known I-V characteristics, for the purpose of calibrating earth solar simulator intensities. Solar cells are flown on high-altitude balloons to altitudes of approximately 120,000 ft (36,576 m), where I-V parameters, temperature, and other data is collected. Flights at this altitude are estimated to be within

0.46 percent of AM0, determined by comparison of the ratio of atmospheric pressure at altitude to that at sea level computed with the Air Research and Develpment Command (ARDC) model of the atmosphere. Helium-filled balloons are flown so as to reach and remain at altitude from two hours before solar noon until two hours after solar noon. The standard solar cell assemblies are mounted on a tracking system which maintains orientation with the sun. Data is transmitted to a ground station during the flight. Upon completion of the mission, a valve is remotely opened and the balloon begins a controlled descent. The test array and equipment are recovered after landing. This method of data collection benefits from the elimination of uncertainties and inaccuracies in measurements, and minimization of corrections which must be made to data taken at lower altitudes. Only two corrections are required with the high-altitude balloon method of cell calibration, one for cell temperature and one for earth-sun distance. Both of these factors are precisely known. Once the reference cell is placed in a simulator, intensity adjustments must be made to match the simulator intensity to that experienced by the cell at altitude. Some cells have been reflown on subsequent flights for correlation of previous data. Repeatability of within ± 1 percent was achieved, verifying the accuracy and validity of previous reference data. [Ref. 2: p.11.3-1]

Until 1985 there had been some question as to the validity of balloon-calibrated solar cells. There was still a question as to the effect atmosphere above the balloon had on the solar radiation spectrum. If this effect was significant, this method of solar cell calibration would not produce the desired accuracy in earth solar simulators. In the summer of 1985, cells flown on a balloon were flown and tested on a space shuttle flight. Comparison of the independent data from the two methods correlated to within one percent [Ref. 3: p.542]. Thus, the high-altitude balloon method has proven to be an accurate method for solar cell calibration. However, as new cells with new spectral characteristics are developed, new standards are required.

E. RADIATION

There are a variety of variables that affect the performance of solar cells in the space environment. Temperature, time, material composition and hardening mechanisms must all be considered in the deployment of a solar array. However, the single greatest effect on an array in space is radiation, which causes performance degradation during the life of a satellite. Damaging radiation is composed of energetic or fast massive particles. Such particles, electrons, protons, and neutrons, inhabit the space environment, in varying densities, and at various times. Some radiation is a secondary effect of other

phenomena, such as Compton electrons, produced by gamma rays. The Van Allen Belts, the Sun, and deep space are all sources of radiation. The mass, energy, and charge of these particles, or associated particles, may interact with or damage solar cells in a number of ways. The radiation phenomena of interest here are ionization and atomic displacement. [Ref. 4: p.3-2]

Ionization occurs when orbital electrons are removed from an atom or molecule. Radiation may affect solar cell materials by several ionization-related effects. The darkening of solar cell coverglasses is an example of one of these effects. Ionizing radiation excites orbital electrons which, upon entering the conduction band, become trapped by impurity atoms, creating defect complexes within the material [Ref. 4: p.3-2].

A large fraction of energy is lost when fast electrons or protons collide with absorbing solar cell atoms. Silicon atoms are displaced from their lattice structure positions by these fast particles, causing permanent degrading damage. The displaced atoms undergo other reactions and ultimately form stable defects which significantly modify equilibrium carrier concentrations and minority carrier lifetimes.[Ref. 4: p.3-3]

It is possible to characterize solar cell damage in terms of changes in minority diffusion length. This method has been widely used, but there are practical and fundamental limitations to this approach. Low energy protons, while causing considerable displacement damage within the junction region of a solar cell, increasing I_o and decreasing V_{oc}, do not change the cell diffusion length [Ref. 4: p.3-18]. In addition, accurate measurement of cell output parameters is much less difficult than measurement of diffusion length, particularly after proton irradiation. Empirical analysis has shown that I_{sc} changes with a linear function of the logarithm of the fluence [Ref. 4: p.3-18]. The variation of solar cell V_{oc} after irradiation has also been empirically related to a logarithmic function. Thus, solar cell damage is generally reduced to the quantifiable changes in I_{sc}, V_{oc}, and maximum power.

The wide range of electron and proton energies present in space have necessitated a method of describing the effects of various types of radiation environment which can be reproduced in the laboratory. Damage equivalent radiation fluence was developed to allow description of the degradation of unshielded silicon solar cells which had experienced 1 MeV electron irradiation under laboratory conditions, and reduce the effects of the space radiation environment on a shielded silicon solar cell to a damage equivalent fluence of 1 MeV electrons in the laboratory [Ref. 4: p.3-24].

Much data has been collected concerning the effects of 1 MeV electron irradiation on solar cells. Particle acceleration, x and gamma radiation, etc. have been utilized and carefully measured to define relative damage effectiveness on solar cells in an effort to simulate with 1 MeV electron radiation, the state of damage that would be experienced in the space environment by an equivalent fluence. This concept has also been extended to the effects of proton irradiation, a more complex problem [Ref. 4: p.3-29]. The degradation of solar cells irradiated with protons is more complex because of the nonuniform nature of the damage, particularly by those with energy below 3 MeV. Proton damage is more severe than that of electrons, but can be normalized to the damage produced by electrons. With this information, simulation of the space environment has been almost the sole method through which solar cell parameter degradation is measured.

F. IN SITU TESTING

Despite successes in simulating the space environment and the modeling of space radiation, the degrading mechanisms which affect spacecraft are still not fully understood. For example, recent research at the Naval Research Laboratory (NRL) indicates that radiation dose rates can have as great or greater impact than overall radiation doses on particular solar cell degradation processes [Ref. 5].

Rarely has the long term process and effects of space radiation been observed. Simulations on Earth are relatively short, and the results analyzed after the fact. The Navigation Technology Satellites 1 and 2 (NTS-1,2), were launched in support of the NAVSTAR Global Positioning System (GPS) in 1977. The GPS program was concerned with the development of high-efficiency solar cells sufficiently radiation resistant to deliver adequate power throughout the mission lifetime of the GPS satellites. Experimental solar cell arrays were on board NTS-1 and NTS-2. These arrays were composed of Si and state-of-the-art GaAs solar cells which were to be evaluated for performance and radiation resistance in the space environment. Information collected during the mission was compared to pre-launch data. It was acknowledged in this experiment that in situ observation was more valuable than simulation tests [Ref. 6: p.1234]. I-V measurements were taken on entire arrays and telemetered to a ground station. Individual cell performance was not observed. This is an important point since the current output of a string of cells is limited to that of the weakest cell in the string. Thus, a defective or damaged cell will cause inaccurate conclusions based on resulting data.

The Living Plume Shield II (LIPS-II), launched by the NRL in February, 1983, carried three double-sided solar panels of Si and GaAs cells. This was a cooperative program by the U.S. Navy and Air Force to build, test, and qualify a GaAs solar panel in space. The GaAs solar cells flown were mounted in three parallel strings of 100 cells. Each string was 25 cells in series by 4 cells in parallel [Ref. 7: p.1108]. Figure 2 outlines the satellite structure and the GaAs array.

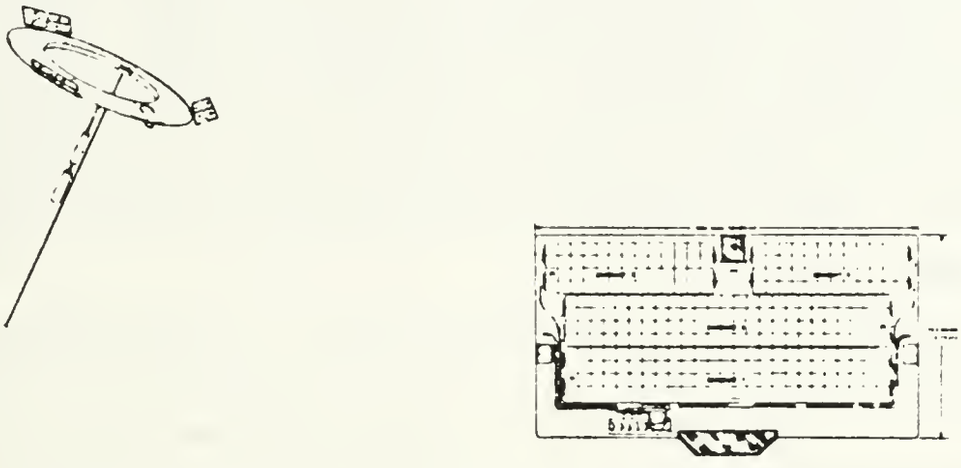


Figure 2. LIPS-II satellite and GaAs solar cell panel. Ref. 7: p.1111

During the first 30 days of operation of the satellite, a 7.3 percent power loss was experienced. These first 30 days of operation were also unmonitored due to satellite orientation problems. While the loss has never been explained, it is believed that the mechanical failure of a solar cell or contact was the cause [Ref. 7: p.1109]. Had individual cells, as well as an array, been tested, the question of this power loss might have been resolved. Further, an autonomous data collection and storage system might also have provided insight into that power loss.

Study of the effects of solar cell annealing as a method of power recovery in degraded solar cells is an ongoing effort. However, complete analysis of such effects requires exhaustive study due to the wide variations possible in temperature and annealing rates, cell power recovery, recovery rates, and the effects these have on the various materials used in solar cell technology. Space environment tests would aid in the understanding and exploitation of this effect.

The Combined Release and Radiation Effects Satellite (CRRES) Program is designed to complete a variety of experiments, among which are the measurement of radiation dose degradation effects in the space environment, and the update of static Earth

radiation belt models [Ref. 8: p.1]. A GaAs solar cell panel experiment on board CRRES will measure the performance characteristics of differently configured solar cell strings and simultaneously measure radiation species (protons, electrons, ions), their flux levels, and energy distributions. Annealing processes and optimum configuration for solar cell panels operating in a high radiation environment will be studied [Ref. 8: p.43]. This work will update existing static radiation belt models based on data collected in the mid 1960's which lacked information on ion species, and pitch angles. This new information will also provide the basis for the first dynamic radiation belt models [Ref. 8: p.6]. Information collected will be used to optimize solar cell panel design criteria in consonance with space radiation measurements [Ref. 8 : p.45]. This emphasis on in situ testing and data collection is an indicator of the importance of this kind of information to satellite designers and users.

Another application of in situ solar cell and array data is in the monitoring of $P_{mp.}$. This is the designed operating point of a solar array. During the life of a spaceborne array $P_{mp.}$ will shift, robbing the satellite of the maximum possible power available from its solar array. Monitoring solar cell performance would provide the opportunity for operational adjustment of the power system on a spacecraft, and more efficiently utilize the remaining power production capabilities of the system.

G. THE MICROPROCESSOR-BASED TEST SYSTEM

Previously, the testing, data collection, data storage, and telemetry of data from a spacecraft to a ground station posed numerous problems. The weight and complexity of required testing devices was limiting. Data storage and handling equipment was bulky. Today, with modern digital techniques and microprocessor controlled devices, these problems have been resolved. Indeed, the capability to collect more and more complex information in space and transfer it to Earth has grown by orders of magnitude. New technologies have miniaturized components to very small weight, volume, and power parameters. The testing and monitoring of solar cells and arrays in space is now a viable option, as has been demonstrated by programs such as those listed above. Based on a simple electronic circuit, one microprocessor-based solar cell array test system, for use in the space environment, is presented below.

II. A NOVEL SOLAR CELL TEST DEVICE

A. APPLICATION

Dr. Sherif Michael and Robert Callaway developed a simple circuit for the measurement of a solar cell I-V curve at the Naval Postgraduate School in Monterey, California, in 1986. This photovoltaic test circuit was designed to facilitate the autonomous testing of individual solar cells, although configuration for strings of cells is also possible. Information accumulated from a number of cells would provide statistically relevant data for accurate assessment of the behavior of an entire array. While this approach precludes use of the cells for power supply, there are benefits to this method. The failure or degredation of a single cell, which can invalidate the data from a string of cells, can be observed and resulting data discarded if inconsistent with the rest of a test array. Such information might have provided some insight into the degradation observed during the first month in orbit of the LIPS-II satellite [Ref. 7: p.1108].

The autonomous operation of this circuit with a controlling system and memory device would provide real-time data acquisition, as on the LIPS experiments [Ref. 9: p.688]. However, if real-time collection is not possible, or undesirable, data storage in bubble memory or other nonvolatile memory devices is possible. The lack of a data recording system created problems in data handling and collection during and after the Solar Cell Calibration Experiment (SCCE) carried out on the space shuttle in 1983-4 [Ref. 10: p.301]. Data storage also allows collection of data for extended periods of time, such as on the proposed CRRES solar cell experiments [Ref. 8: p.10].

Data for entire I-V curves can be collected, opposed to a few points, as on the LIPS tests, where only seven data points were collected per curve [Ref. 7: p.1108]. Since the $P_{mp.}$ point shifts during the life of an array, monitoring this parameter is important. A few points of data will not provide accurate enough information for analysis of such deviations. Parameters such as temperature, sun angle, time of day, etc., can also be stored with cell I-V data, simplifying retrieval and correlation of environmental information.

B. TEST CIRCUIT REQUIREMENTS

There were a number of requirements in the development of a low power, lightweight, inexpensive, and accurate solar cell parameter measurement scheme capable of

operating in the space environment. Below are specifications developed for the autonomous test circuit [Ref. 11: p.68].

1. Minimize series resistance through current sinks.
2. Ability to record data accurately.
3. Capability to sweep current through entire I-V curve, (V_{oc} to zero voltage at I_{sc}).
4. Capability to measure a series of multiplexed cells and sensors accurately.
5. Internal resistance of multiplexer CMOS switches should not affect measured data.
6. Circuit must be simple and small .
7. Low power and low thermal output.
8. Buffer input and output signals to insure accuracy.

C. DESIGN

The actual solar cell biasing circuit is composed of a high gain (h_{fe}) bipolar junction transistor (BJT), (eg. 2N3405, with $h_{fe} > 400$), placed in a common emitter configuration [Ref. 11: p.70]. The test solar cell is placed between a 5 volt power source, V_{ps} , and the collector of the transistor. This voltage level was chosen to preclude the possibility of saturating the transistor in the circuit. A lower voltage would not ensure this. A resistor is placed between emitter and ground. An operational amplifier, in a unity gain configuration, utilizes its high input impedence to buffer and prevent undesirable current-produced effects in resulting test circuit data. Figure 3 on page 13 depicts this circuit [Ref. 11: p. 71].

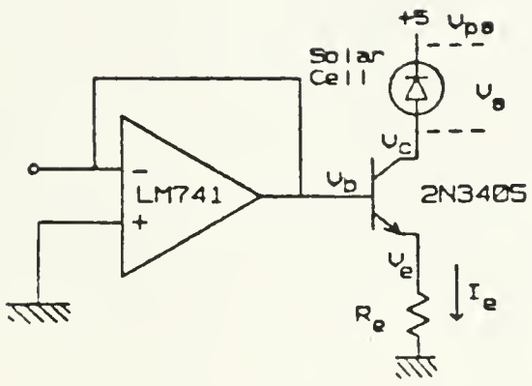


Figure 3. Novel solar cell biasing circuit.

The test cell provides a load to the transistor. While very little or no current, (I_b), is allowed to flow into the base of transistor, collector current, (I_c), approximates emitter current, (I_e). In this situation, with the cell illuminated and no voltage applied to the base of the transistor, the voltage across the solar cell is V_{oc}. The difference between V_{ps} and collector voltage V_c, is the solar cell circuit voltage:

$$V_c - V_{ps} = V_s$$

Emitter voltage, V_e , divided by the emitter resistance, R_e , provides circuit current, I_s, and

$$V_e/R_e = I_e = I_s$$

Since there is no appreciable current drain, I_e and, thus, I_s are approximately zero. This provides one endpoint of the cell's I-V curve. As a voltage is applied to the transistor base, further forward biasing the device, V_c decreases, and current drain increases, until the voltage across the solar cell drops to zero and I_{sc} is reached. By stepping input voltages to the transistor base, data points for an entire curve may be collected by measuring V_c and V_e. [Ref. 11: p.68]

A multiplexed system was tested with a set of these circuits attached to a counter circuit, which simulated microprocessor control, and a digital-to-analog device. The capability to produce a large number of data points through the D/A converter resulted in very accurate I-V curves.

III. MICROPROCESSOR CONTROLLER

A. SYSTEM DESIGN

Rather than design a controller system from scratch, a search was performed to identify and acquire an operative controller that fulfilled the needs envisioned in the deployment of the solar cell measuring system. The system needed to be capable of operation in the space environment. Low power consumption, simplicity, small size, and compatibility with the measurement circuits were also necessary. Assembly language programming was initially assumed, due to available supporting hardware at NPS, but this requirement was relaxed to allow for a higher level language with the addition of new compilers to NPS.

The controller designed for operation of the NPS Autonomous Space Shuttle Payload Bay Launch Vibro-acoustics Experiment was ultimately chosen for the solar cell measurement system. The vibro-acoustics experiment, an ongoing project, was designed for flight in the payload bay of a space shuttle to measure the vibration and acoustic effects experienced by the space shuttle during the stresses of a launch. The experiment requires a NASA-approved autonomous control system to detect shuttle launch, execute a power-up sequence, and operate the experiment. The controller also monitors the progress of the experiment and contains diagnostics within its software. By necessity, characteristics desirable for the solar cell measurement system were inherent in this device. The controller was well developed and documented, both in hardware and in software. The microprocessor system was compatible with typical assembly languages for which support at NPS was readily available. The addition of a 'C' language compiler and subsequent programming of the controller in 'C' was a further incentive in the selection decision. The controller hardware had been developed for low power consumption, as well as minimal size. The entire controller, including memory, was placed on a 9 x 5.5 inch board and required a single 10 volt power supply. An external RS-232 cable provided terminal access for diagnostics. I/O ports end at 44-pin connectors for easy attachment of external devices. The vibro-acoustics experiment also developed the use of bubble memory as a means of nonvolatile data storage. This capability was not chosen for use with the solar cell measurement project due to cost and the continued development of this capability within the vibro-acoustics project. However, bubble memory presents a viable option for future inclusion as a data storage device with the

solar cell measurement system. A complete schematic diagram is included in Appendix A.

B. CONTROLLER COMPONENTS

1. NSC800 Microprocessor

The heart of the controller is the National Semiconductor NSC800 microprocessor. This device provides the advantages of CMOS construction, a small heating coefficient, and low power consumption. The processor has the ability to multiplex the address/data bus. An 8-bit machine, the NSC800 can be operated in a 16-bit address format by multiplexing lower address lines (A0-A7), latching them externally, and combining them with the upper non-multiplexed address bus (A8-A15), which creates an effective address space of 64K [Ref. 12: p.8]. This family of devices has a number of compatible peripheral devices and is capable of addressing multiple input/output (I/O) devices. The microprocessor supports the Z-80 assembly language instruction set.[Ref. 13: p.17]

2. NSC810A RAM-I/O-Timer

The National Semiconductor NSC810A is a random access memory (RAM), timer, and I/O peripheral device. This is another CMOS machine, which incorporates 1024 bits of built-in static RAM in an 8-bit format. The I/O section has 22 programmable bits arranged into three programmable ports. Port A, composed of 8 bits, is capable of basic I/O operation, or one of three strobed modes. Port B is operable only in a basic I/O mode. Port C can be used for basic I/O or as a handshake in conjunction with port A operation as a programmable timer [Refs. 13: p.26, 14: p.1]. Through individual port bit manipulation, external devices may be operated. Designed for operation with the NSC800, two of these devices are utilized in the controller system.

3. IM6402 Universal Asynchronous Receiver Transmitter (UART)

The UART provides the controller the ability to interface with a terminal, which, in turn, allows troubleshooting and diagnostic operation of the system. The UART must also transmit parallel data from the controller data bus to external serial data lines. The INTERSIL IM6402 generates the clocking for transmitter and receiver operation for such asynchronous interfacing. This UART is another low power CMOS device. [Ref. 13: p.43]

4. MM58167 Real Time Clock

The vibro-acoustics experiment required initiation of experiments at a particular time. Power-up and power-down were also part of a power conservation requirement

within the completely autonomous experiment. The ability to operate in this fashion is also compatible with the operation of the solar cell measurement system which need consume power only when collecting data, excepting memory devices. The clock features a four year calendar with month to thousandths of a second selection. The chip includes a programmable alarm circuit for power-up and power-down commands. The MM58167 is a CMOS device manufactured by National Semiconductor. [Refs. 13: p.51, 15: p.1].

5. Memory

a. EPROM

Driver memory for the controller is composed of standard CMOS UV erasable PROMs. The 2764 series EPROM is a low power, high performance device with good noise immunity. This memory chip has a standard pin configuration and a variety of versions for specialized applications, including wide operating temperature ranges [Ref. 16: p.1]. The controller board has space allocated for up to eight memory chips. The current solar experiment configuration utilizes five of these spaces for EPROMs, which provide the operating code for the controller. EPROMs provide an inexpensive method for rapid software development and experimental investigation into the limits of controller operation.

b. Bubble Memory

A nonvolatile memory was required for data storage on the vibro-acoustics experiment. A similar memory system is also necessary for solar cell measurement data storage. Bubble memory provides a relatively low power, megabit capacity which is nonvolatile, even when power is removed, purposely or in the event of a failure [Ref. 13: p.31]. These features make this format ideal for large quantities of data and long term storage, as might be experienced during a space mission. Bubble memory characteristics also facilitate the retrieval of data for transmission at extended intervals. However, incorporation of bubble memory, under development for the vibro-acoustics project, was eliminated at the current time to preclude any delays in the solar cell project which might occur from this ongoing development. The expense of bubble memory, in conjunction with currently available research funds, also precluded inclusion as a storage device in this project.

c. RAM

A simple alternative to bubble memory is a battery powered static RAM system, which may remain powered at all times. This approach would provide for an

inexpensive nonvolatile memory with small weight and power penalties. The current controller is not configured for a separate memory power circuit. It was decided to leave this relatively simple modification for later addition and concentrate on the data collection, storage, and retrieval system for the solar cell project. Thus, static RAM, standard 8K word, 8-bit chips were chosen for data storage. The 6264 machine is an industry standard with high-speed and low power characteristics [Ref. 17: p.1]. The static RAM requires no refresh and dissipates less power than dynamic RAM.

IV. SOLAR CELL ARRAY TEST CIRCUIT

A. OVERVIEW

The solar cell array test circuit was designed to provide a bias on test array solar cells and collect the resulting information relating to individual cell voltage output and current. This required conversion of digital signals to an analog form to facilitate biasing the transistors used in the novel cell test circuits. The two signals tapped from each solar cell were then reconverted to digital form and passed to the controller microprocessor for manipulation and storage.

B. COMPONENTS

1. DAC0800 8-bit Digital-to-Analog Converter

The DAC0800 is a standard, 8-bit CMOS digital-to-analog converter, which provides low power consumption and a 100 ns output current settling time. It requires little direction or external control and operates under a wide power supply range. A wide range of applications and compatability with standard CMOS and TTL devices also made it appealing.[Ref. 18: p.1]

2. HI-506A Analog Multiplexer

The HI-506A is a rugged analog multiplexer with the capability to automatically multiplex or demultiplex analog signals. That is, it is manufactured with the necessary internal switches so as to be insensitive to signal flow direction. For this application the demultiplexing capability was required. This component was also designed for space use and has a high electrostatic discharge (ESD) resistance. [Ref. 19: p.1]

3. ADC0809 A/D Converter and Multiplexer

The ADC0809 device incorporates an 8 channel analog multiplexer, aiding in the minimization of individual hardware devices, and allowing direct access to analog signals. The analog-to-digital converter is an 8-bit machine using successive approximation for conversion. The converter requires no external scale adjustments. Latched address inputs and outputs also maximize ease of interface with microprocessors. It is advertised to have no missing codes and a total unadjusted error of $\pm$ 1/2LSB, important considerations in the accuracy of the final output values. A CMOS device, it also provides high speed, accuracy, minimal temperature dependence, and low power consumption. [Ref. 20: p.1]

C. DIGITAL-TO-ANALOG CONVERSION AND DEMULTIPLEXING

The 8-bit digital bias signal generated in the controller was directed to the DAC0800. (Figure 4 on page 21 refers.) The input was converted to positive current output and referenced to ground. The output current signal was converted through an LM741 op-amp to a positive low impedance voltage output. This output was then forwarded to the HI506A Analog Mulitplexer. Individual solar cell address information was wired into the multiplexer from NSC810-1. The received voltage signal was thus routed to the correct cell on the array.

D. ANALOG TO DIGITAL CONVERSION AND MULTIPLEXING

The desired analog voltage signal was actually the difference between source voltage and transistor collector voltage. An LM741, in an instrument amplifier configuration, was placed across the biased solar cell to provide this difference voltage. The resulting desired analog cell output voltage was then routed to an ADC0809. Here, address lines from Port B of NSC810-1 facilitated demultiplexing. The signal was converted to a digital signal and passed to Port A of NSC810-2, as may be seen in the solar cell array test circuit schematic in Figure 4 on page 21.

Current information was tapped from the biasing transistor's emitter lead in analog voltage form. It may be recalled that this voltage value, divided by the known emitter resistance of 10 ohms, provides the desired current value. This computation is accomplished by software in the microprocessor after analog-to-digital conversion, prior to retrieval of data from storage. In a manner similar to that of the voltage signal convergence scheme, the current information signal was directed through a second ADC0809 and the resulting digital data placed on Port B of NSC810-2.

E. INTERFACE

The two NSC810 I/O devices were utilized for digital signal output, chip and data control, and digital data input. Port A of NSC810-1 was used for bias signal output. The stepped output signal provided a bias for the solar cell novel test circuit. The lower three bits of Port B on NSC810-1 carried solar cell address information. The fourth and fifth bits provided a 'start convergence' pulse for analog-to-digital conversion on the ADC0809 chips. The 'output enable' signal to the ADC0809 devices were always asserted. Signals output by the NSC810 ports were internally latched, providing simplification of timing and allowance for the settling time of signals before reading or conversion.

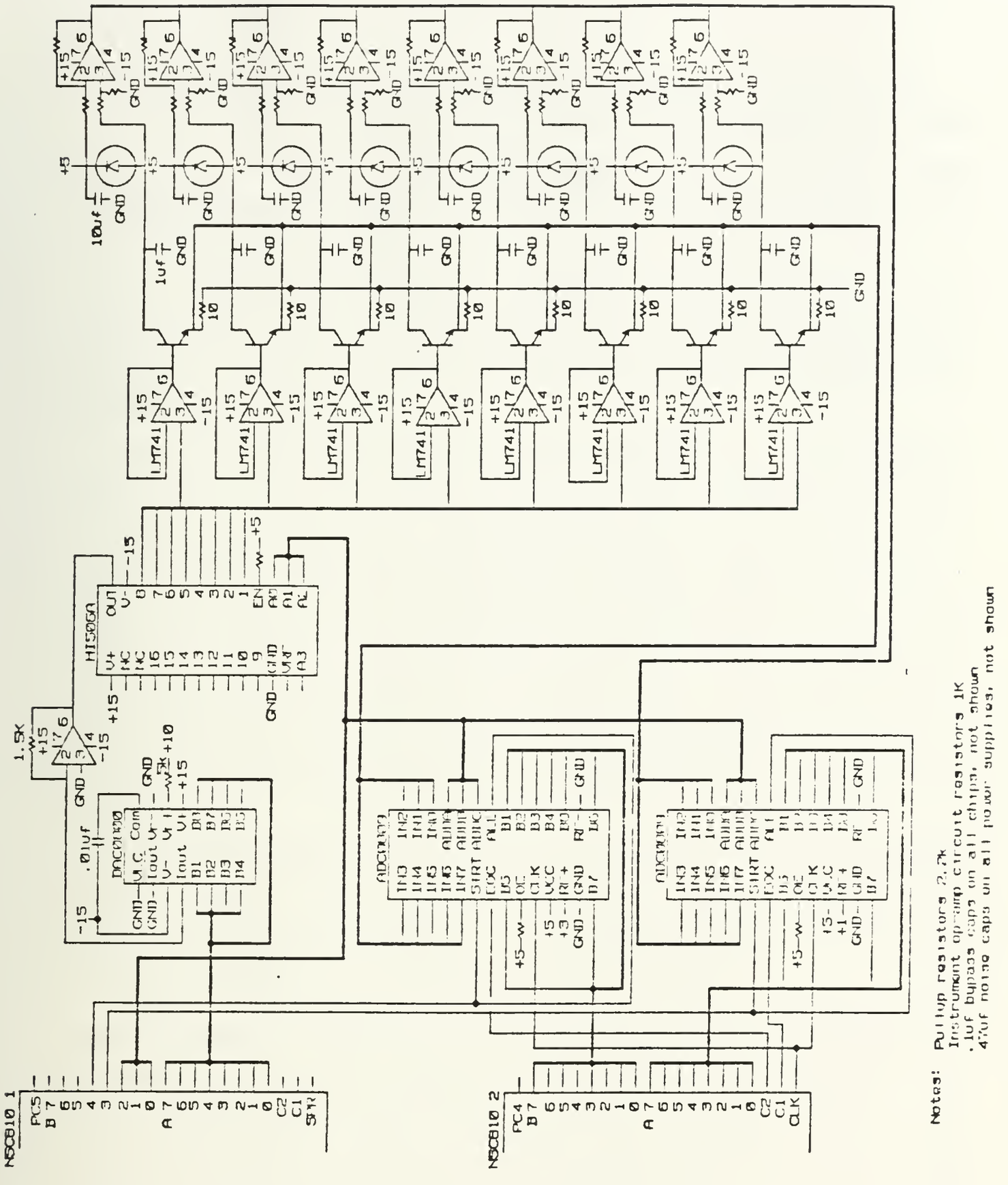


Figure 4. Solar cell array test circuit schematic.

Two of the ports on the NSC810-2 were used for test circuit data retrieval. Port A received voltage information, and Port B current information. Two bits of Port C were utilized to record the end-of-convergence signal from the ADC0809s. A clock signal from the controller was passed to the ADC0809s via a spare port bit on the NSC810-2.

V. SOFTWARE

A. CONTROLLER ROUTINES

The controller software files below are described in a cursory manner. They were originally designed for another project and modified for use in this application. A more complete description and use of the files developed for the NPS vibro-acoustics project is available in, *Control of an Experiment to Measure Acoustic Noise in the Space Shuttle,* by Charles Cameron [Ref. 21].

The software developed for the microprocesser controller was written to provide for autonomous operation of the system. This required a timer and alarm routine to power up and power down the system. A substantial diagnostics routine and menu further provided for ease in manipulation and testing of the controller. Some of these features were incorporated in the portions of code utilized for the solar cell test system. The 'C' language was chosen largely for its readability, opposed to assembly languages. While a "high level" language, C provides the ability to simply manipulate individual bits, as well as operate on words and bytes. C is also very portable. Appendix B includes the start-up and operating routines for the microprocessor controller. These files are, for the most part, modified versions of those written by Cameron [Ref. 21: Chapter 4 and Appendix B].

Header files, designated by the ".h" in the file name, are used to define and declare variables, constants, functions, routines, structures, etc. which will be utilized in the overall program by various modules of code. The header files indicate where externally defined code is located. This is necessary for program compilation.

The header file solareva.h defines parameters necessary for start-up and operation of the controller. Bit definitions, I/O assignments, and clock routine definitions are provided. Some of these have been renamed or modified for specific use in the solar cell test routines, which utilize I/O in a different manner than originally intended for the vibro-acoustics routines. The files solar.h, initial.h, convert.h, global.h, inout.h, delay.h, newio.h, and clock.h are all header files which declare functions, variables, etc., used within the associated C file, which has the same name as the header file. Such header files must be included with routines which utilize these "externally" defined parameters.

C files, those whose name are followed by ".c", are the actual start-up, operation, and test routines compiled and executed by the microprocessor. With the exception of

celltest.c, all the .c files are copies of, or modified versions of the files written by Cameron for the vibro-acoustics project [Ref. 21: Appendix B].

Initial.c is the initialization code for the controller. The operations executed here set I/O ports, initialize functions and sequences, and start the timer operation. The initial.c file is executed by solar.c. The solar.c module provides monitor and keyboard interface, displays the version of the routines used, and prints a menu for routine operation, testing, and diagnostics. The separate modules which actually accomplish these actions are accessed by solar.c. The importance of header files becomes apparent here as routines external to this file are required for its execution. Solar.c contains the "main" portion of the program, the code from which all other routines are accessed, and to which they ultimately return. The menu selections 'Execution' and 'Data Memory' were added for experiment execution and data retrieval, respectively, in the menu section of code.

The convert.c module provides ASCII to hexidecimal, decimal, or binary-coded decimal (bcd) conversions, as well as the reverse operations. This is necessary for human readability at the monitor, and keyboard interface.

Inout.c provides the actual data output and execution of keyboard input commands. While the functions in this file are all written in C, some functions were more efficiently written in assembly language, and hence, newio.s includes the operations of input and output of data to and from I/O ports. Note that the .s indicates an assembly language module. The start.s file is the processor initialization code which is executed when the system is reset or initially powered up.

The file global.c includes information necessary for the timer and alarm routines defined in clock.c. The clock.c file provides for initialization of the clock and setting a wakeup time via a menu driven routine. The clock operates on a four year cycle and may be set from months to seconds of accuracy and waketime. Delay.s creates an "n" millisecond time delay. This delay was used at various points during software and hardware interface to check completion or execution of digital-to-analog and analog-to-digital functions as well as verify conversion time. A symbol table has been included which specifies variable definitions and declarations within compiled routines. The table also provides memory address information, storage allocation, and total memory and addressing necessary for programming PROMs.

Programming was done on an IBM-PC utilizing MS/DOS. Compilation was accomplished with the Uniware C Compiler, produced by the Software Development Sys-

tems Co., of Downers Grove, Illinois. Completed programs were linked, assembled, and transferred to EPROMs on the same machine.

B. SOLAR CELL ARRAY ROUTINE

The solar cell array test routine, celltest.c, was designed to directly interface the test circuit with the controller, input/output information, and manipulate that information for storage and use. The entire program is included in Appendix C.

Celltest.c first defines the variables used in the routine which assign constants for bit manipulation during execution of the program. ARRAYSZ defines the number of solar cells in the array. STOP and START provide high and low assertion for operating bits on the ADC0809s. Variable declarations are also made prior to entry into the executable code.

Three structures, groupings of specific variables that may be handled together in a particular format, are defined. The first, PORT1_B, allows bit operation on an output port to assign the solar cell address of interest and to provide the 'start convergence' pulse necessary for analog-to-digital conversion by the hardware, all via the 'command' operator. The second structure, C_PORT, provides access to two bits which must be read when the ADC0809s have completed the convergence cycle. The third structure is called data_pt. This grouping assigns two variables which will hold information for a single data point. Each pair of these points is assigned to a cell in the array, experiment_data, which is defined to allow memory space for a maximum of 256 data points for each of the eight cells included in the array. Figure 5 on page 26 follows the flow of this routine.

Following these definitions and declarations, the executable portion of the routine begins. It is labeled 'execute'. The routine begins with a loop for which each iteration completes the testing of one solar cell. Following the 'for' statement which initiates this loop are three statements which initialize a variable counter and two comparison variables. After these initializations are two statements which address a particular solar cell via the PORT1_B structure through a 'command' statement, and execute the assignment by output through port 1 of NSC810-1, the addressing and control output port.

A second 'for' statement is next executed, providing for the biasing voltage ladder and associated control, manipulation, and storage of resulting data for each data point created. There are 256 voltage levels, evenly distributed through a three volt reference source, which provide the steps in this loop. These values are passed, one at a time, through the D/A converter, analog demultiplexer, buffer op-amp, and to the cell biasing

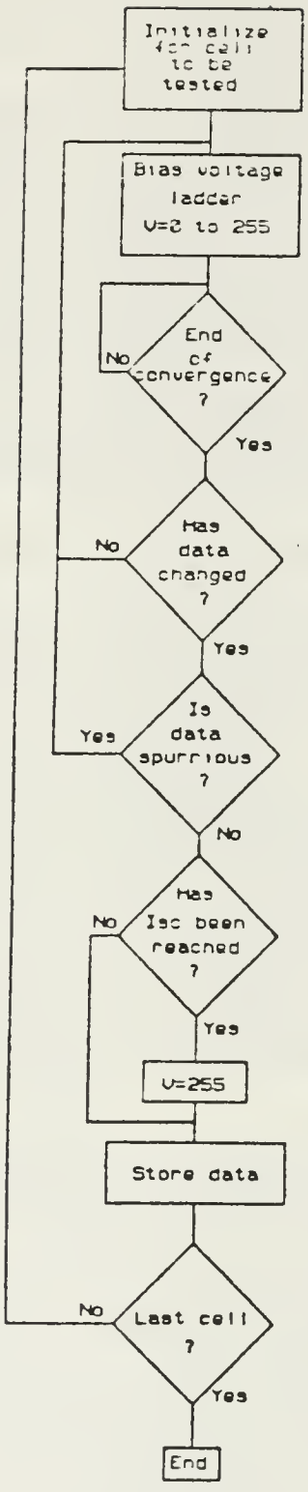


Figure 5. 'Execute' routine flow diagram.

circuit. Once the cell has been biased, the start convergence pulse, 'strtcn', is asserted with START and STOP statements. A single pulse is used to start the convergence cycle of both A/D converters. When each ADC0809 has completed convergence of its assigned analog signal to a digital signal, 'end of convergence' pulses are transmitted by the analog-to-digital converters. These pulses are received and latched by two bits of port C on NSC810-2. A 'while' construct waits until these corresponding bits have been asserted before allowing execution to continue, thus assuring complete conversion before storage of data.

The next 'if' statement checks for a current value greater than zero and deletes unchanged data values. This prevents unnecessary storage of data or storage while the input voltage ladder overcomes the forward voltage of the biasing transistor, and during which, no current flows. The voltage ladder provides voltage that will eventually saturate the biasing transistor and cause a negative voltage measurement. It should be noted that the analog-to-digital conversion recognizes only magnitude and not gender, preventing a simple search for negative values. A succeeding nested pair of 'if' statements provides a smoothing effect on data by deleting data which is inconsistent with the curve as a result of conversion or other error. The I-V curve of interest need not collect data beyond the point where voltage has reached zero. At this point the difference between the bias voltage and the solar cell output has become zero and short circuit current has been achieved. Thus, the succeeding 'if' statement ends the the input ladder by incrementing the counter to 255. The two digital data, voltdata and currentdata, are stored in RAM at this point. The storage statements reflect the fact that not every step of the biasing loop will result in storage of a data point; the 'row' parameter only increments when data is stored into the array. Figure 6 on page 28 follows the flow of this routine.

The succeeding routine, 'retrieve', is executed by selection of the appropriate choice on the controller menu. When called upon, this routine retrieves the stored data from RAM by first identifying the appropriate cell number, entered via the input terminal. This cell number corresponds to the column in the storage array which holds the stored data. The retrieving loop is written to stop retrieval at the end of data in the loop.

Voltage and current data were stored in hexidecimal form to minimize storage allocation requirements. Thus, for plotting and easy human interpretation, the data must be converted to a decimal form. Data is converted to floating decimal upon retrieval and appropriate scaling factors applied. A one volt reference was used in the analog-to-digital conversion of the voltage values. This corresponds to 0.0039 volts per step

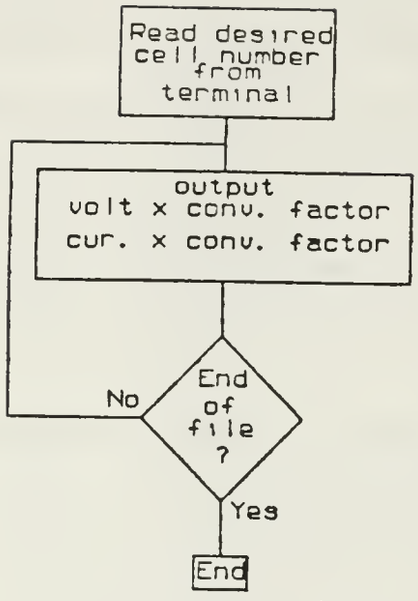


Figure 6. 'Retrieve' routine flow diagram.

through the 256 step ladder within the ADC0809, the applied multiplication factor used during the voltage conversion step. However, inherent conversion error requires an offset factor which increased the multiplication factor to .0040 for silicon cells. A wider range of current values was required because of the different outputs produced by Si and GaAs solar cells. Thus, a larger reference voltage was required for the ADC0809 used in converting the voltages which indicated current levels. The three volt reference used here corresponded to .0117 volts per each of the 256 steps. However, this value required a further division by ten ohms, to complete the conversion from a voltage value to a current value. In order to account for the conversion error of this ADC0809, the factor was changed to 0.0116. The emitter resistors must also be accurately measured prior to current calculations and accounted for as well. Finally, the two decimal form values are output by the microprocessor. These values, with a third parameter unique to the plotting routines used for data display, were transferred to floppy disk via the Procomm interfacing program. Header and trailer inputs were included for convenience interfacing an existing plotting routine in another program.

VI. TEST AND RESULTS

A. TEST

Data output format was a manipulation of hexidecimal data into floating decimal. This facilitated interface with a 'personal computer' in the laboratory and the transfer of data to another storage medium for easy analysis. Conversion from hexidecimal to floating point decimal need not be accomplished within the controller, but was in this case for convenience. Data was retrieved from the solar cell test system via a commercial interface program. Both available and versatile, 'Procomm', developed by Datastorm Technologies, Inc., was chosen for this task. Retrieved data was transferred to floppy disk files and printed out in graphic form via a plotting routine on the solar laboratory computer. The plotting routine was part of the program designed for solar cell data collection, storage, and analysis using the NPS Solar Simulator and associated hardware developed by Don Gold [Ref. 22: Appendix D].

Ideally, the solar cell array test system was designed to allow data collection from an entire array of cells. Practically, the system was limited by the solar simulator light source used at NPS with this project. The illuminated area produced by the simulator provides for, at most, a pair of 2 cm square cells under AM0 conditions. Thus, data collection was limited to single cells.

B. RESULTS

Data collected with the microprocessor-based system for silicon solar cells was plotted and compared to that produced by the direct measurement and storage system in place in the NPS Solar Laboratory. Figure 7 on page 31 and Figure 8 on page 32 are of two different 2 cm x 2 cm silicon solar cells and show the similarities in the results of the two methods of data collection. A sample table of silicon data is provided in Appendix D.

The results for GaAs solar cell comparisons is somewhat different, apparent in Figure 9 on page 33. While a small adjustment of approximately 60μV was added for offset and resistor precision error, this adjustment proved inappropriate for GaAs cells. At this point, the effects of several types of analog-to-digital conversion errors should be investigated more closely in conjunction with the conversion process. These effects include [Ref. 23: p.113.]:

1. Offset error values which are within the range equivalent to the LSB but have shifted the range upwards, effectively extending the error range.
2. Gain error caused by an input value that is a fractional value of the full scale range (FSR), resulting in a corresponding fractional binary output. The binary output becomes detached from its analog input with greater fractional values of FSR. Figure 10 on page 34 portrays a relationship between practical and ideal transfer curves of binary representations for fractional FSR values.
3. Nonlinearity for the range of analog voltages applied when compared to the binary codes produced. If this error is significant, differential linearity may cause a skip of certain binary codes, known as "missing codes".

Tests executed with the delay of one or more milliseconds indicated that ample time was allowed for complete convergence, and thus, should not be a factor in the errors observed. Tests conducted without a delay on silicon cells produced no variations, compared to those with the delay.

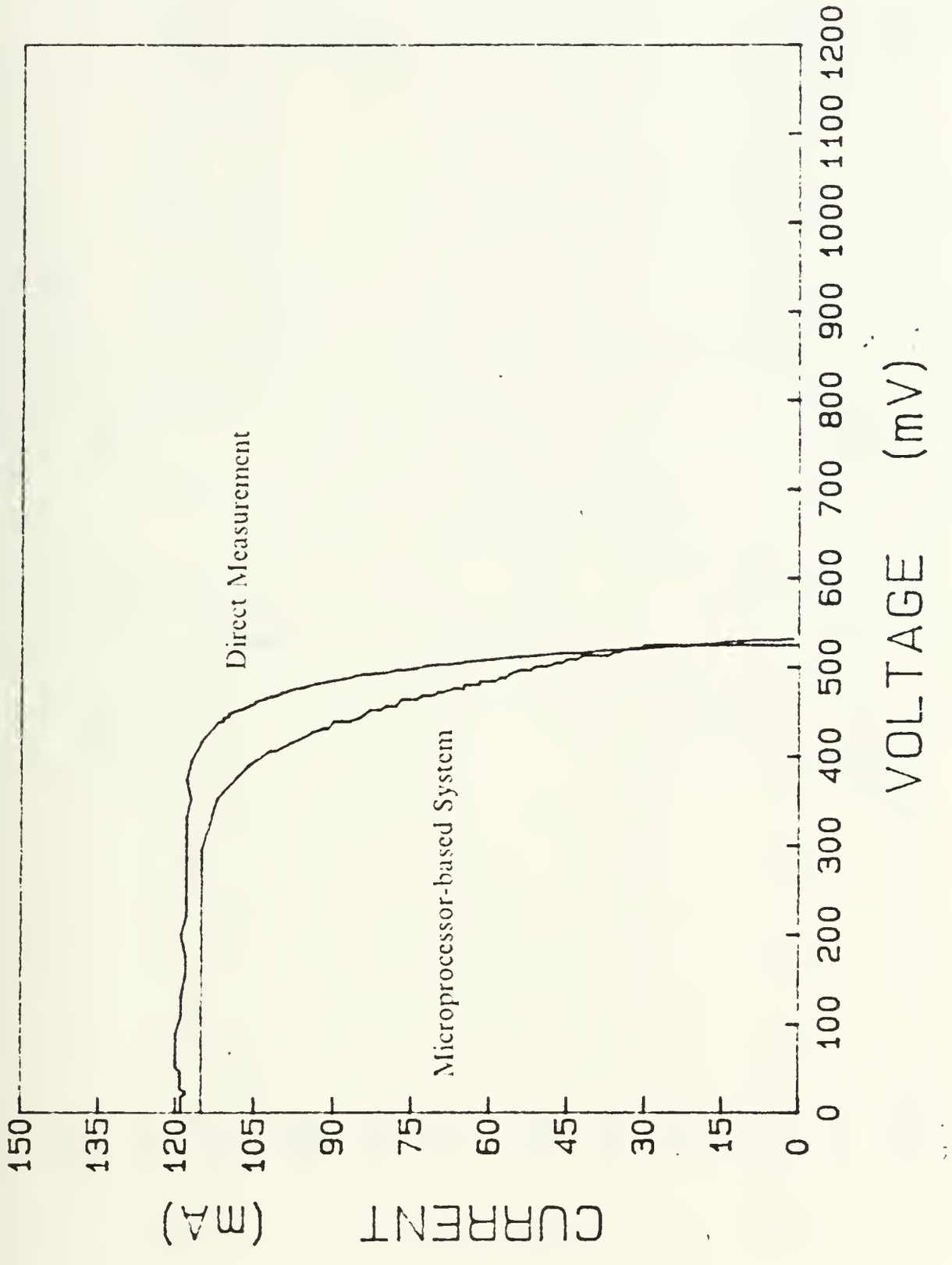


Figure 7. Sample 1, silicon solar cell I-V curves.

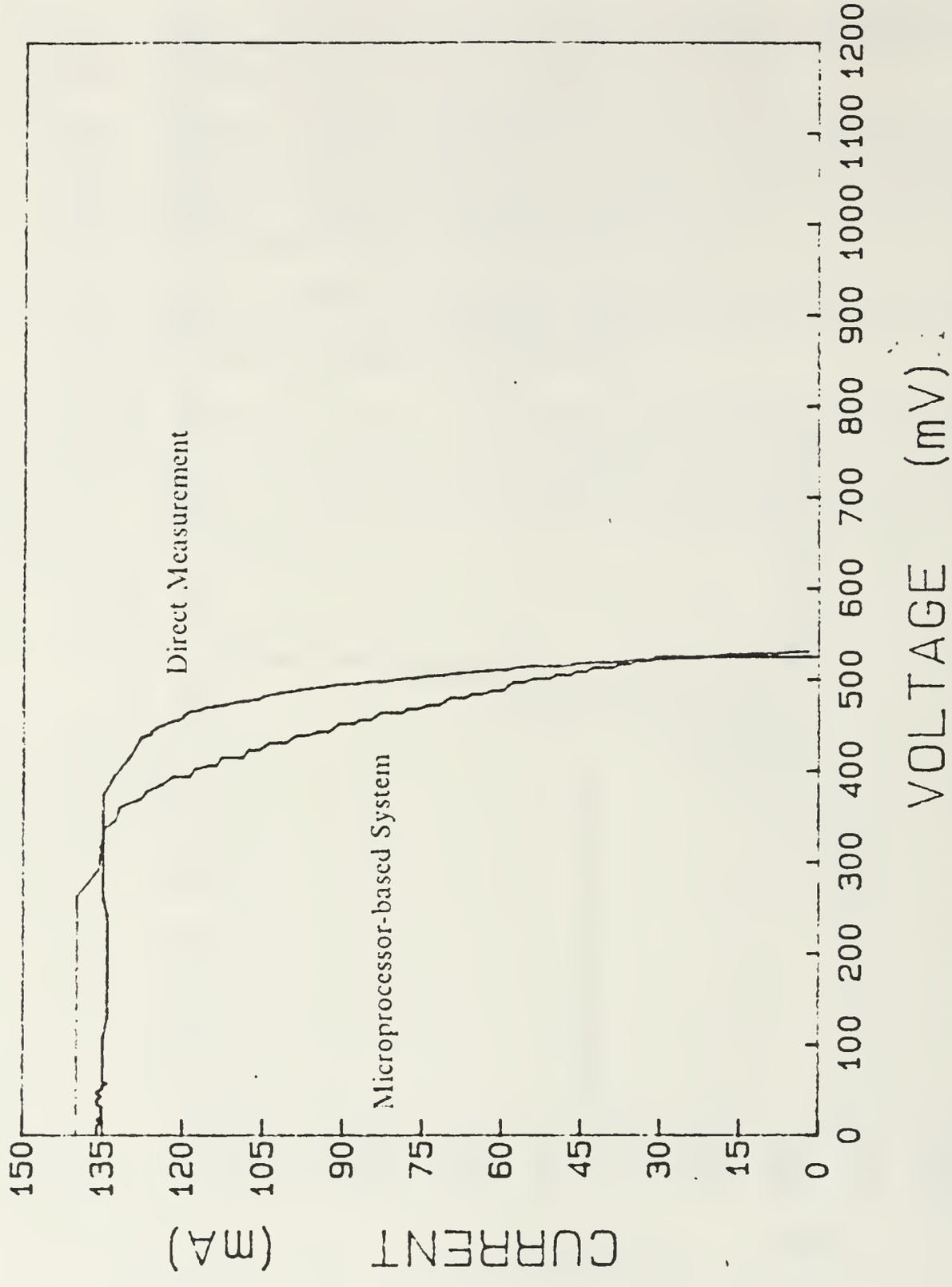


Figure 8. Sample 2, silicon solar cell I-V curves.

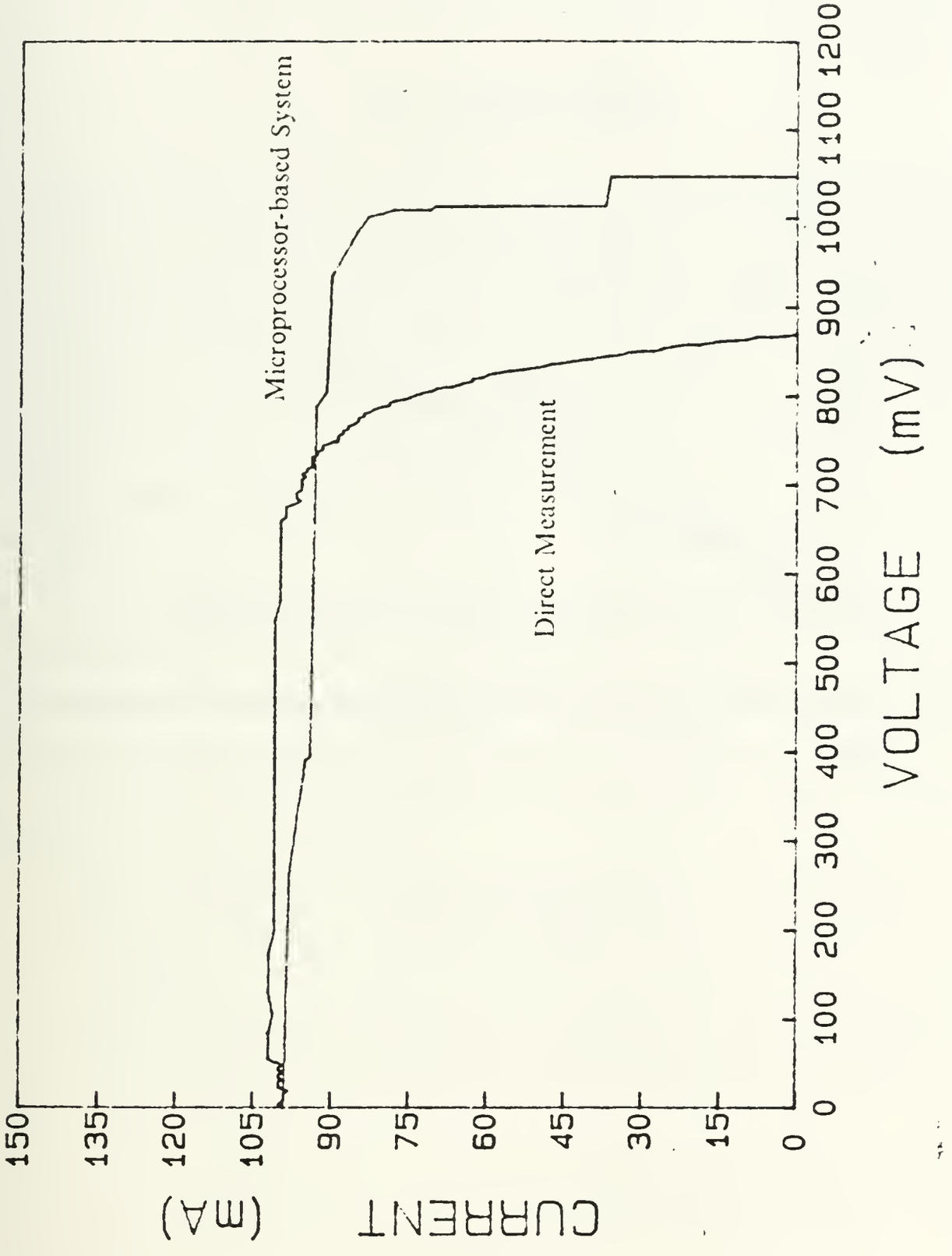


Figure 9. Sample gallium-arsenide solar cell I-V curves.

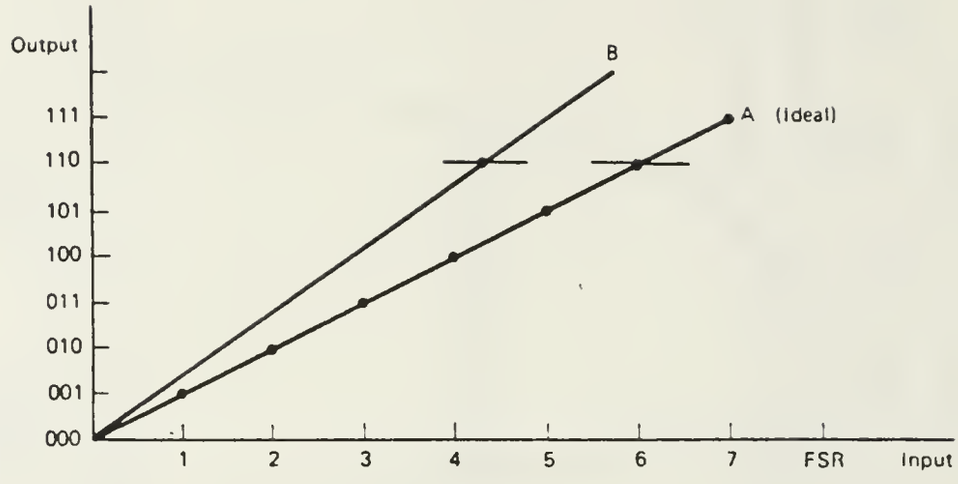


Figure 10. Practical versus ideal binary/fractional FSR transfer curve. [Ref. 23: p.113]

VII. CONCLUSIONS

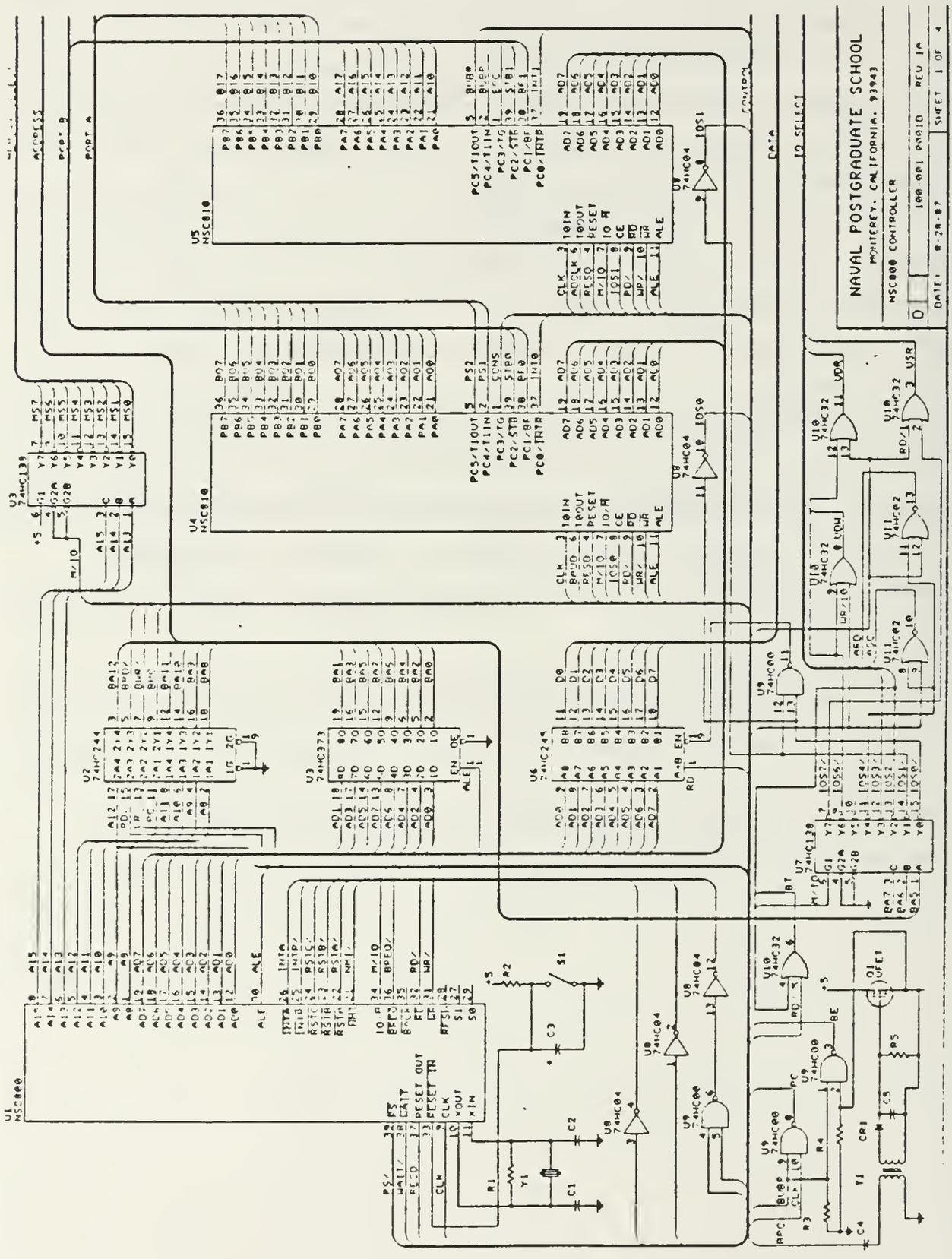
The microprocessor-based solar array I-V measurement system was built using a controller previously designed and tested at the Naval Postgraduate School in Monterey, California. This controller's programming was modified, and its I/O ports connected to circuitry specifically designed for this project. Digital biasing signals were demultiplexed through an addressing scheme and converted to analog voltages. These voltages were then used to bias a novel solar cell biasing circuit, from which two voltage taps were read on each cell. Successive taps, representing cell voltage and current data points, were multiplexed, converted to digital values, and stored in controller memory; data representing a complete solar cell I-V curve for each cell in the test array. Another programmed routine enabled retrieval of this data, manipulated into decimal form for handling and analysis.

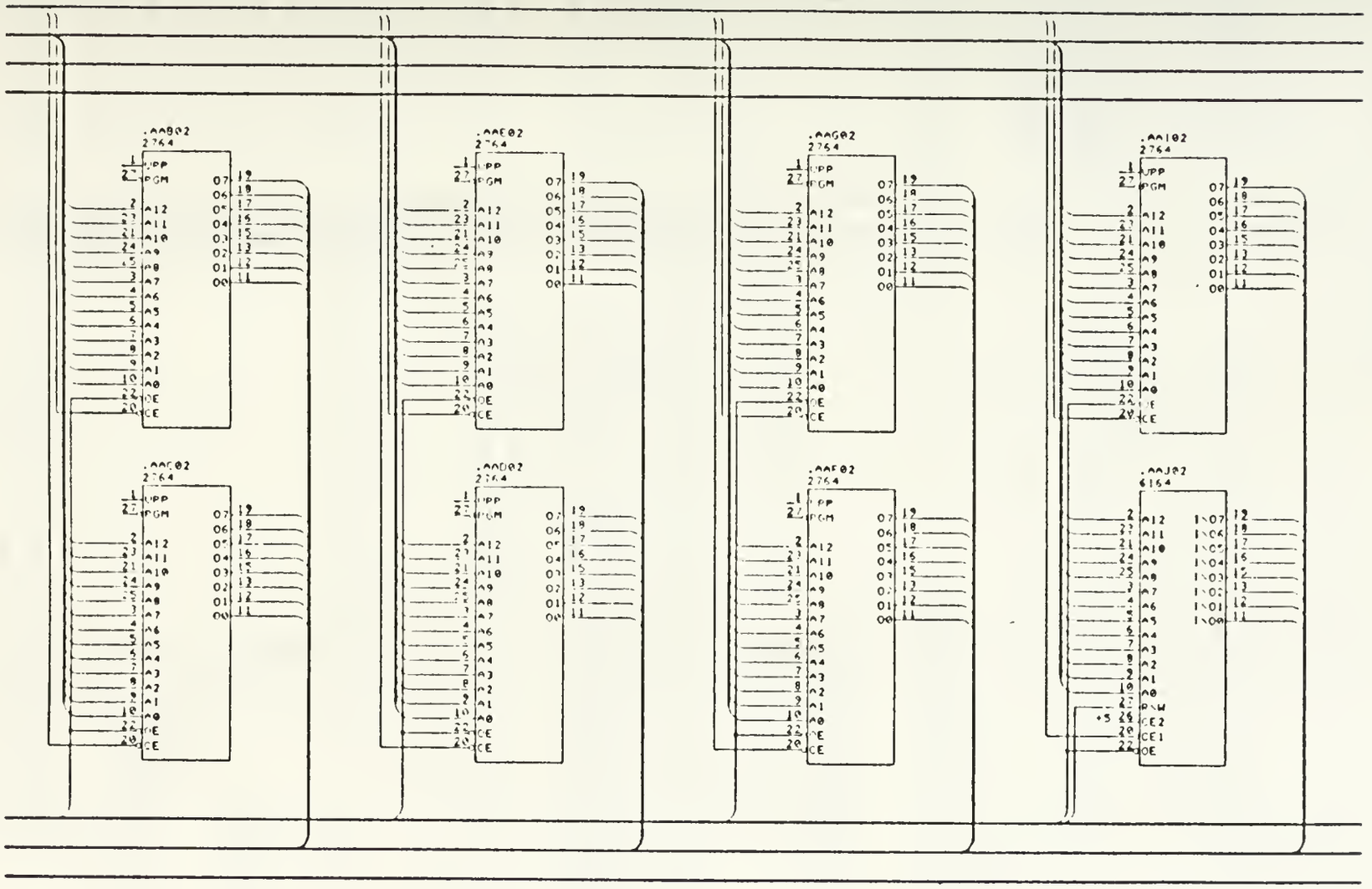
The microprocessor-based solar array parameter measurement system is a viable method for collection, storage, and retrieval of I-V information and other pertinent data. The system is capable of accurately measuring a number of cells in an AM0 environment. Data may be accessed from system memory, manipulated, and analyzed.

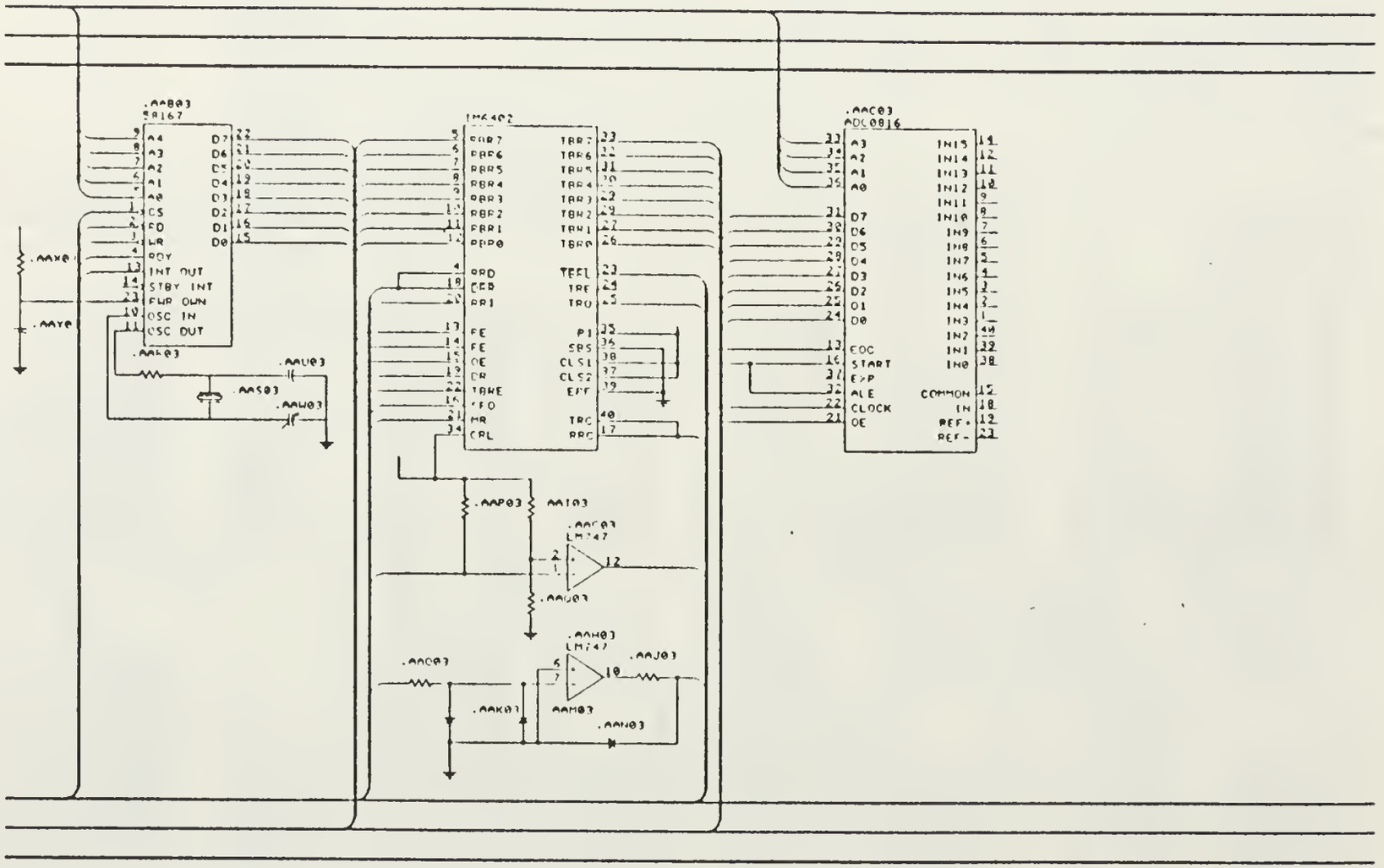
There are a number of improvements and possible avenues of study to pursue in the further development of this project.

1. A more complete study of the accuracy of the systems output, including the possibility of using a 16-bit microprocessor system.
2. A software or hardware approach for analysis and compensation for conversion error effects.
3. Installation of the alarm clock system for timed power-up and power-down.
4. The inclusion of nonvolatile memory; battery powered or bubble memory.

APPENDIX A. NPS MICROPROCESSOR CONTROLLER SCHEMATIC




.AA802
2764
.AA902
2764
.AAA02
2764
.AAB02
2764
.AAC02
2764
.AAD02
2764
.AAE02
2764
.AAE02
2764
.AAG02
2764
.AAE02
2764
.AA102
2764
.AAJ02
6164
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA, 93943
NSCAA0 CONTROLLER
100-001-00010
REV 1A
DATE: 8-20-87
SHEET 1 OF 4



NAVAL POSTGRADUATE SCHOOL		
MONTEREY, CALIFORNIA, 93943		
NSC800 CONTROLLER		
D	100-001-0001D	REV 1A
DATE: 8-28-87	SHEET 1 OF 4	

APPENDIX B. CONTROLLER START-UP AND OPERATING CODE

A. FILENAME SOLAREVA.H

```
/*       May 4, 1988      solareva.h                */

#define TRIES     3       /* Number of times to try something before giving up. */
#define  STRLEN   7       /* Number of characters to allow for integer
                            characters, including a null terminator. */
#define HSTRLEN 2 /* Number of characters to allow for hexadecimal
                            characters*/
#define HEXINTSTRLEN 4    /* Number of characters in a hexadecimal word. */
#define DUMPWIDTH 16      /* Number of bytes in a line of a memory dump. */

#define TERMON    0x08    /* Points to the terminal connection line in NSC810 #1,
                            Port C, Pin 3. */

/* Bit definitions for port C of NSC810 #2. (Base address is 0x22.)
         Bit #            Meaning
         5                X
         4                X
         3                X
         2                End of Convergence signal  EOC-2
         1                End of Convergence signal  EOC-1
         0                X
*/
#define READC1    0x02    /* Points to the NSC810 #1, Port C, R/W register. */
#define BCLRC1    0x0a    /* Points to the NSC810 #1, Port C, Clear register. */
#define  BSETC1   0x0e    /* Points to the NSC810 #1, Port C, Set register. */
#define BCLRC2    0x2a    /* Points to the NSC810 #2, Port C, Clear register. */
#define BSETC2    0x2e    /* Points to the NSC810 #2, Port C, Set register. */

#define MDR1      0x07    /* See the documentation for a description of the */
#define DDRA1     0x04    /* use of these ports. */
#define DDRB1     0x05
#define DDRC1     0x06
#define TM01      0x18
#define TOLB1     0x10
#define TOHB1     0X11
#define START01   0x15
#define MDR2      0x27
#define DDRA2     0x24
#define DDRB2     0x25
#define DDRC2     0x26
#define TM02      0x38
#define TOLB2     0x30
#define TOHB2     0x31
#define START02   0x35

#define PRTDATA   0xc0    /* Port number for data from RS-232C interface. */
#define PRTCTRL   0xe0    /* Port number for control information from RS-232C
```

```
                              interface. */
#define PRTOUTRDY 0x01  /* Bit zero of the PRTCTRL byte is a one if the printer
                              is ready to accept data and zero otherwise. */
#define PRTRDY  0x02    /* Bit one of the PRTCTRL byte is a one if there is
                              data to be read and zero otherwise. */

#define PORT1_DA      0x00      /* D/A Output address (port A on NSC810-1) */
#define PORT1_CTRL    0x01      /* D/A Control Port (port B on NSC810-1) */
#define PORT2_ADV     0x20      /* A/D Voltage input */
#define PORT2_ADC     0x21      /* A/D Current input */
#define PORTC2        0x22      /* A/D EOC port */

#define TRUE       0xff
#define FALSE      0x00
#define ASCII      0        /* Used as a parameter to showbubbuff(). */
#define HEX        1        /* Used as a parameter to showbubbuff(). */
#define NULL     0x00       /* The following are ASCII definitions. */
#define BELL       0x07
#define SPACE    0x20
#define DELETE   0x7f


#define THOUSANDTHS         0x60     /* The ports for reading the date and time. */
#define HUNDREDTHS          0x61
#define SECONDS             0x62
#define MINUTES             0x63
#define HOURS               0x64
#define WEEKDAY             0x65
#define DATE                0x66
#define MONTH               0x67

struct datetime {           /* This structure contains binary coded */
          char     month;           /* decimal data as defined for the National */
          char     date;            /* Semiconductor MM58167A Microprocessor */
          char     hour;            /* Real Time Clock. */
          char     minute;
          char     second;
          char     hundredths;
          char     thousandths;
};

struct idatetime {                  /* This structure contains the same */
          int      imonth;          /* information as the datetime structure, but*/
          int      idate;           /* in integer format.  clockint() takes care */
          int      ihour;           /* of converting from BCD to integer format. */
          int      iminute;
          int      isecond;
          int      ihundredths;
          int      ithousandths;
};
```

B. FILENAME SOLAR.H

```
/* April 19, 1988  solar.h  */

extern void version(void);
extern void memory_dump(void);
extern char menu(void);
```

C. FILENAME INITIAL.H

```
/*    April 19, 1988  initial.h  */

extern void inithardware(void);
```

D. FILENAME CONVERT.H

```
/*  convert.h  April 20, 1988  */



extern char atoh(char *ascii);
extern unsigned int atohexint(char ascii[]);
extern int atoi(char *s);
extern char *bcd_asc(char bcd);
extern int bcd_int(char bcd);
extern char *ctoh(char byte);
extern char int_bcd(int decimal);
extern char *itoa(int n, char s[]);
extern char tolower(char c);
extern char *uitoh(unsigned int word);
```

E. FILENAME GLOBAL.H

```
/*   April 19, 1988   global.h   */

extern char prtconnected;

extern struct datetime clock;

extern struct idatetime waketime;
```

F. FILENAME INOUT.H

```
/* April 19, 1988  inout.h */

extern char checkprt(void), gethex(void), termin(void);
extern int getint(void);
extern unsigned int gethexint(void);
extern void dump(unsigned int address, unsigned int length);
extern void echo(char data), portdump(char *string);
extern char termin(void);
extern void testinput(void), testoutput(void);
```

G. FILENAME DELAY.H

```
/*************************************************************************/
/*delay.h    May 19, 1988    Header file for delay.s in ASMSOURCE directory */

extern void delay(int);
```

H. FILENAME NEWIO.H

```
/* April 20, 1988  newio.h
  header for newio.s, in ASMSOURCE Directory.          */

extern char input(char port);
extern void output(char port, char data);
```

I. FILENAME CLOCK.H

```
/* This file contains external declarations in prototype format for
all the functions defined in "clock.c". */

extern void clockint(struct datetime *clock,struct idatetime *iclock);
extern void clockread(struct datetime *your_clock);
extern char clockcompare(struct idatetime *clock1,struct idatetime *clock2);
extern void clockset(struct datetime *clock);
extern void clocksum(struct idatetime *result,struct idatetime *clock1,
                struct idatetime *clock2);
extern void dump_clock(struct datetime *clock);
extern void rtc(void);
extern void show_waketime(struct idatetime *waketime);
extern void testtimeout(void);
extern char timeout(int delaytime,int measure);
```

J. FILENAME INITIAL.C

```
/*       May 20, 1988              initial.c                          */
/* Baud rates: TOLB1 bit is 0x07 = 9600, 0x0f = 4800, 0x1f = 2400, 0x3f = 1200,
   0x7f = 600, 0xff = 300 */

#include "newio.h"
#include "solar.h"
#include "solareva.h"

void inithardware(void);


void inithardware(void)
{
         output(MDR1,0x00);              /* Mode byte 810 #1. (basic I/O) */
         output(DDRA1,0xff);             /* Set port A to output. */
         output(DDRB1,0xff);             /* Set port B to output. */
         output(DDRC1,0x30);             /* Set port C to input/output. */
         output(TMO1,0x00);              /* Stop the timer. */
         output(TMO1,0x25);              /* Set timer mode. */
         output(TOLB1,0x1f);             /* Set low byte for timer. (Baud rate)*/
         output(TOHB1,0x00);             /* Set high byte for timer. */
         output(START01,0x07);           /* Start timer. */
         output(MDR2,0x00);              /* Mode byte for 810 #2. */
         output(DDRA2,0x00);             /* Set port A to input. */
         output(DDRB2,0x00);             /* Set port B to input. */
         output(DDRC2,0x31);             /* Set port C to input/output. */
         output(TMO2,0x00);              /* Stop the timer. */
         output(TMO2,0x25);              /* Set timer mode. */
         output(TOLB2,0x0a);             /* Set low byte for timer. */
         output(TOHB2,0x00);             /* Set high byte for timer. */
         output(START02,0x0a);     /* Start timer. */
         output(BCLRC2,0x30);      /* Set bits in port C */
}
```

K. FILENAME SOLAR.C

```
/*        April 11, 1988          solar.c                  */

#include "solareva.h"
#include "convert.h"
#include "inout.h"
#include "initial.h"
#include "global.h"
#include "clock.h"

extern void execute(void);
extern void retrieve(void);

void version(void);
void memory_dump(void);
char menu(void);

/*************************************************************************/
void version(void)
{
         portdump(
"\n\rBob Oxborrow's Control Program for Solar Panel Research.\n\r");
         portdump("Version 1.00  May 9, 1988\n\r\n\r");
}

/*************************************************************************/
/* This routine lets the user produce memory dumps for any section of memory.*/
void memory_dump(void)
{
         unsigned int address;     /* Will hold the starting address of the dump.*/
         unsigned int length;      /* Will hold the number of bytes to dump.*/
         while (TRUE) {
                  portdump("Please specify address:  ");
                  address = gethexint();
                  portdump("\n\rPlease specify number of bytes to dump (0 to quit):  ");
                  length = gethexint();
                  if(length == 0)
                           break;
                  dump(address,length);
         }
}
/*************************************************************************/
char menu(void)
{
         char data;

         while(TRUE) {
                  portdump("\n\rSolar panel evaluation control program.\n\r\n\r");
                  portdump("A  Real time clock functions.\n\r");
                  portdump("B  Memory dump.\n\r");
        portdump("C  Execution.\n\r");
        portdump("D  Data Memory.\n\r");

                  data = termin();
                  echo(data);
```

```
                portdump("\n\r");
                switch (data) {
                        case 'a':  case 'A':
                                rtc();
                                break;
                        case 'b':  case 'B':
                                memory_dump();
                                break;
          case 'c':  case 'C':
              execute();
              break;
          case 'd':  case 'D':
              retrieve();
              break;



                default:
                                portdump("Use a valid letter please! \n\r");
                }
        }
}

/*****************************************************************************/
void main(void)
{
        inithardware();
        if (prtconnected = checkprt()) {
                version();
                menu();
        }
}
```

L. FILENAME CONVERT.C

```
/*       April 11, 1988          convert.c                */

#include "solar.h"
#include "newio.h"
#include "solareva.h"

char atoh(char *ascii);
unsigned int atohexint(char ascii[]);
int atoi(char *s);
char *bcd_asc(char bcd);
int bcd_int(char bcd);
char *ctoh(char byte);
char int_bcd(int decimal);
char *itoa(int n, char s[]);
char tolower(char c);
char *uitoh(unsigned int word);

/*************************************************************************/
/* This routine converts a two-byte ASCII string representing a valid
   hexadecimal byte into a single hexadecimal byte. */
/*************************************************************************/
char atoh(char *ascii)  /*A string representing a hexadecimal byte. */
{
         int     i;
         char    result; /* The hexadecimal byte after conversion. */

         result = 0;
         for (i=0;i < HSTRLEN && ascii[i] != NULL;++i) {
                 result *= 16;
                 if ( '0' <= ascii[i] && '9' >= ascii[i])
                         result += ascii[i] - '0';
                 else if ('a' <= ascii[i] && 'f' >= ascii[i])
                         result += 10 + ascii[i] - 'a';
         }
         return(result);
}

/*************************************************************************/
/* This routine converts a four-byte ASCII string representing a valid
   hexadecimal word into a single unsigned integer. */
/*************************************************************************/
unsigned int atohexint(char ascii[])
{
         int     i;
         unsigned int result;     /* The hexadecimal word after conversion. */

         result = 0;
         for (i=0;i < HEXINTSTRLEN && ascii[i] != NULL;++i) {
                 result *= 16;
                 if ( '0' <= ascii[i] && '9' >= ascii[i])
                         result += ascii[i] - '0';
                 else if ('a' <= ascii[i] && 'f' >= ascii[i])
                         result += 10 + ascii[i] - 'a';
         }
```

```
        return(result);
}

/*************************************************************************/
int atoi(char *s)       /* convert string to integer */
{
        static int n, sign;
        sign = 1;
        n = 0;
        switch (*s) {
                case '-': sign = -1;
                case '+': ++s;
                }
        while (*s >= '0' && *s <= '9') n = 10 * n + *s++ - '0';
        return(sign * n);
}


/*************************************************************************/
/* Convert a byte of binary coded decimal data to character string format. */
/* No check is made to ensure that input data really IS in BCD format. */
char *bcd_asc(char bcd)    /* Tested March 16, 1987 */
{
        static char ascii[3];
        int bcdint;

        bcdint = 0x00ff & ((int) bcd);            /* Convert to integer. */
        /* If the tens digit is a zero, put a blank in its place;
           otherwise, put an ASCII digit there. */
        ascii[0] = (0xf0 & bcdint) ?
                (0x30   (bcdint >> 4)) : ' ';
        ascii[1] = 0x30   ((bcdint & 0x0f));      /* Get the units digit. */
        ascii[2] = NULL;                  /* Terminate the string with
                                                  a null. */
        return(ascii);
}

/*************************************************************************/
/* Convert a byte of binary coded decimal data to integer format. */
/* No check is made to ensure input data really IS in BCD format. */
/*Tested March 16, 1987 */
int bcd_int(char bcd)     /* The BCD character to be converted. */
{
        int bcdint, result;
        /* Take the units by masking off the tens. */
        /* Then throw away the units and keep
           the tens.*/
        bcdint = 0x00ff & (int) bcd;
        result =  0x000f & bcdint;
        /*Multiply the tens by 10, and add to result.*/
        result += 10 * (bcdint >> 4);
        return(result);
}

/*************************************************************************/
/* Convert a character to hexadecimal ASCII string format. */
```

```
char *ctoh(char byte)
{
        static char ascii[HSTRLEN];
        int byteint, nibble, base;

        byteint  = 0x00ff & ((int) byte);       /* Convert to integer. */
        nibble = byteint >> 4;                  /* Get the tens digit. */
        /* Find out whether the nibble is in the range [0-9], in which
           case its ASCII representation starts at 0x30 (48 decimal), or
           [10-15], in which case the ASCII representation starts at
           A = 0x41 (65 decimal).  In the latter case, add the value of the
           nibble to 65-10 = 55. */
        base = (nibble >= 10) ? 55 : 48;
        ascii[0] = base + nibble;
        nibble = byteint & 0x0f;          /* Get the units digit. */
        base = (nibble >= 10) ? 55 : 48;
        ascii[1] = base + nibble;
        ascii[2] = NULL;                  /* Terminate the string with
                                                  a null. */
        return(ascii);
}

/*************************************************************************/
/* This routine converts an integer to a binary coded decimal character.
   Since 99 is the largest legitimate BCD number, the argument "decimal"
   is taken modulo 100. */
char int_bcd(int decimal) /* The number to be converted. */
{
        int result;

        /* Make sure decimal is a positive number. */

        decimal = (decimal < 0) ? -decimal : decimal;
        decimal %= 100;                   /* If decimal is too big, take
                                             it modulo 100. */
        result = (decimal / 10) << 4;     /* Get the tens and shift them into the
                                             high order half of the byte. */
        result += decimal % 10;           /* Add in the units. */
        return((char) result);
}

/*************************************************************************/
/* itoa - convert n to characters in s. */
char *itoa(int n, char s[])
{
        static int c, k;
        static char *p, *q;

        if ((k = n) < 0)
                k = -k;
        q = p = s;
        do {
                *p++ = k % 10 + '0';
        } while (k /= 10);
        if (n < 0) *p++ = '-';
        *p = 0;
```

```
        while (q < --p) {
                c = *q; *q++ = *p; *p = c; }
        return (s);
}

/*****************************************************************************/
/* tolower  -  if the input is in [A..Z], convert to lower case */
char tolower(char c)
{
        if ('A' <= c && c <= 'Z')
        return (c + 0x20);
        return c;

}

/*****************************************************************************/
/* Convert an unsigned integer to hexadecimal ASCII string format. */
char *uitoh(unsigned int word)
{
        static char ascii[HEXINTSTRLEN + 1];
        unsigned int nibble;
        int i;

        ascii[HEXINTSTRLEN] = NULL;
        for (i=0;i < HEXINTSTRLEN;++i) {
                /* Get the current nibble, in order from most to least significant. */
                nibble = 0x000f & (word >> (4 * (3 - i)));
                /* If nibble >= 10, convert it to a letter from 'A' to 'F'.
                   If nibble <  10, convert it to a letter from '0' to '9'. */
                ascii[i] = (nibble >= 10) ? ('A' + nibble - 10) : ('0' + nibble);
        }
        return(ascii);
}
```

M. FILENAME INOUT.C

```
/*        April 20, 1988          inout.c                  */

#include "solar.h"
#include "convert.h"
#include "solareva.h"
#include "global.h"
#include "newio.h"

char checkprt(void);
void dump(unsigned int address, unsigned int length);
void echo(char data);
char gethex(void);
unsigned int gethexint(void);
int getint(void);
void portdump(char *string);
char termin(void);
void testinput(void);
void testoutput(void);

/*************************************************************************/
/* This routine checks to see if there is a printer connected to the
   controller.  It returns TRUE if there is one, FALSE otherwise. */
char checkprt(void)
{
         /* If the TERMON bit of the READC1 port is 0, then a terminal
            is connected. In this case return TRUE; FALSE otherwise. */

/* This is temporary until we get the terminal recognition hardware working. */
/*       return(( input(READC1)) & TERMON);*/
         return(TRUE);
}

/*************************************************************************/
/* This routine produces a hexadecimal dump of any section of memory. */
void dump(unsigned int address, unsigned int length)
{
         unsigned int i;                          /* Points to the current byte being dumped. */
         char ascii[DUMPWIDTH+1];  /* Contains the ASCII equivalent of each byte. */

         ascii[DUMPWIDTH] = NULL;  /* Make sure ascii has a null delimiter
                                                                  to look like a C string. */
         /* Convert length to a multiple of DUMPWIDTH. */
         length = ((length + DUMPWIDTH-1)/DUMPWIDTH) * DUMPWIDTH;
         for (i=0;i<length;i++) {
                  if (0==i%DUMPWIDTH) {     /* Dump the ascii version and start a
                                                                  new line every DUMPWIDTH chara
                           if (i > 0) {
                                    portdump(ascii);
                                    portdump("\n\r");
                           }
                           portdump(uitoh(address+i)); /* Also, dump the current address. */
                           portdump(":  ");
                  }
                  /* Put extra spaces in the middle of each line. */
```

```
                if (0==i%(DUMPWIDTH/2) && 0 != i%DUMPWIDTH) {
                        portdump("  ");
                }
                portdump(ctoh(*(char *) (address+i)));     /* Dump each byte individually. */
                portdump(" ");
                /* Insert the current character in the string "ascii".*/
                /* If it's not printable, replace it. */
                ascii[i%DUMPWIDTH] = *(char *) (address+i);
                if (ascii[i%DUMPWIDTH] < SPACE    ascii[i%DUMPWIDTH] >= DELETE) {
                        ascii[i%DUMPWIDTH] = '.';
                }
        }
        /* Make sure ascii is printed again at the end of the last line. */
        if (i > 0) {
                portdump(ascii);
                portdump("\n\r");
        }
}

/*****************************************************************************/
/* Echo a character to the terminal. */
void echo(char data)
{
        char buf[2];
        buf[0] = data;
        buf[1] = NULL;   /* Buf[] ends in a null because it's a C string. */
        portdump(buf);   /* Use portdump() to output the string. */
}

/*****************************************************************************/
/* This routine gets a hexadecimal byte from the terminal.*/
/*****************************************************************************/
char gethex(void)
{
        int i;
        char string[HSTRLEN + 1];

        string[HSTRLEN] = NULL;
        for (i=0;i < HSTRLEN;++i) {
                string[i] = tolower(termin());
                echo(string[i]);
                if (string[i] >= 'a' && string[i] <= 'f')
                        continue;
                if (string[i] >= '0' && string[i] <= '9')
                        continue;
                string[i] = NULL;
                break;
        }
        return(atoh(string));

}

/*****************************************************************************/
/* This routine gets a hexadecimal word (two bytes) from the terminal.*/
/*****************************************************************************/
```

```
unsigned int gethexint(void)
{
        int i;
        char string[HEXINTSTRLEN+1];

        string[HEXINTSTRLEN] = NULL;
        for (i=0;i < HEXINTSTRLEN;++i) {
                string[i] = tolower(termin());
                echo(string[i]);
                if (string[i] >= 'a' && string[i] <= 'f')
                        continue;
                if (string[i] >= '0' && string[i] <= '9')
                        continue;
                string[i] = NULL;
                break;
        }
        return(atohexint(string));

}

/*************************************************************************/
/* Get an integer from the terminal. */
int getint(void)
{
        int i;
        char string[STRLEN];

        string[STRLEN] = NULL;
        for (i=0;i < STRLEN;++i) {
                string[i] = termin();
                echo(string[i]);
                if (string[i] < '0'    string[i] > '9') {
                        string[i] = NULL;
                        break;
                }
        }
        return(atoi(string));
}

/*************************************************************************/
/* This routine sends character strings to the PRTDATA port. */
void  portdump(char *string)
{
        if (!prtconnected)
                return;
        while (*string) {
                /* The terminal is ready when status bit 0 is a one. */
                while(!(PRTOUTRDY & input(PRTCTRL)));
                output(PRTDATA,*string++);

        }
}

/*************************************************************************/
char termin(void)
{
```

```
        while (TRUE) {
                /* Bit 1 will be 1 when data is present. Wait for data. */
                if (input(PRTCTRL) & PRTRDY)
                        break;
        }
        return(input(PRTDATA));  /* Data is present, so read it. */
}

/*************************************************************************/
void testinput(void)
{
        int     port;   /* Port number to be entered from the keyboard.*/
        char    data;   /* Data to be read from that port. */

        portdump("Specify port address to be read (in hexadecimal):  ");
        port = gethex();          /* Get the port address. */
        portdump("\n\r");
        data = input(port);               /* Read from the port. */
        portdump("Data from port (in hexadecimal):  ");
        portdump(ctoh(data));
        portdump("\n\r");
}

/*************************************************************************/
/* This routine outputs a character to a specified port. */
/*************************************************************************/
void testoutput(void)
{
        int     port;           /* The port address. */
        char    data;           /* The data to be sent to the port. */

        portdump("Specify port address to be written to (in hexadecimal):  ");
        port = gethex();          /* Get the port address. */
        portdump("\n\r");
        portdump("Specify the data to be sent to the port (in hexadecimal):  ");
        data = gethex();
        output(port,data);
}
```

N. FILENAME NEWIO.S

```
;         February 19, 1988         newio.s

          export   input, output
          region   code

; char input(char port);
input:
          push ix                   ;There are no local variables.
          ld       ix,0
          add      ix,sp
          ld       c,(ix+4)         ;Put port address in register c.
          in       a,(c)            ;Get the data from the port.
          pop      ix               ;Restore ix to the value it had before this
                                    ;function was called.
          ret

; void output (char port, char data);
output:
          push     ix
          ld       ix,0             ;There are no local variables.
          add      ix,sp
          ld       c,(ix+4)         ;Put port address in register c.
          ld       a,(ix+6)         ;Put data in register a.
          out      (c),a            ;Write the data to the port.
          pop      ix               ;Restore ix to the value it had before this
                                    ;function was called.
          ret
```

O. FILENAME START.S

```
;*********************************************************************
;
;       February 19, 1988                           start.s
;
;       This startup code initializes interrupt vectors and runs START at
;       reset
;       to initialize RAM and call the user function main().
;       The companion link specification file is "spec" which defines
;       many of the imported symbols.  Also see file "mbrk.asm" for the
;       mbrk() function if you want to use malloc() or calloc().
;*********************************************************************
        export  START,MBRKPTR
        import  main,STACKTOP,RAMDATA,ZRAM,ZRAMSZ,IRAM,IRAMSZ,MRAM

;*********************************************************************
;       Define a variable to track memory allocations in mbrk().
;*********************************************************************
        region  ram
MBRKPTR ds      2                       ; (char *) to available memory

;*********************************************************************
;       Reset code must be linked to address 0.
;*********************************************************************
                region  reset
                ld      sp, lo STACKTOP ; initial stack pointer (0x10000 as 0)
                jp      START           ; initial execution address

                org     0x08
ARESTART:       ;RESTART LOCATION 1
                jp      START
                org     0x10
BRESTART:       ;RESTART LOCATION 2
                jp      START
                org     0x18
CRESTART:       ;RESTART LOCATION 3
                jp      START
                org     0x20
DRESTART:       ;RESTART LOCATION 4
                jp      START
                org     0x28
ERESTART:       ;RESTART LOCATION 5
                jp      START
                org     0x2C
FRESTART:       ;RESTART LOCATION C
                jp      START
                org     0x30
GRESTART:       ;RESTART LOCATION 6
                jp      START
                org     0x34
HRESTART:       ;RESTART LOCATION B
                jp      START
                org     0x38
IRESTART:       ;RESTART LOCATION 7
                jp      START
```

```
                org     0x3C
JRESTART:       ;RESTART LOCATION A
                jp      START
                org     0x66
NONMASKI:       ;NON-MASKABLE INTERRUPT
                jp      START

;*************************************************************
;       This code can be anywhere; the reset code jumps to it.
;*************************************************************
        region  code
START   ld      ix,0            ; end of stack frame chain
        ld      hl,MRAM         ; initialize memory allocator
        ld      (MBRKPTR),hl

        ;*************************************************************
        ;               Zero out uninitialized RAM.
        ; It is assumed here that ZRAMSZ > 1 but this is guaranteed
        ; as long as MBRKPTR (above) is defined in region ram.
        ;*************************************************************
        ld      hl,ZRAM         ; zero ZRAMSZ bytes here
        ld      (hl),0          ; zero first byte
        ld      de,ZRAM+1       ; repeatedly zero other bytes
        ld      bc,ZRAMSZ-1
        ldir

        ;*************************************************************
        ;               Initialize other RAM from ROM.
        ;*************************************************************
        ld      hl,RAMDATA
        ld      de,IRAM
        ld      bc,IRAMSZ
        ld      a,b
        or      c
        jr      z,none
        ldir
none:

        ;*************************************************************
        ;               Invoke main() with no arguments.
        ;*************************************************************
        call    main            ; any return value is "int" in de
done:   halt                    ; halt if main returns

;*************************************************************************
;       To vector an interrupt to a C function, you must go though
;       a register save routine like the one shown here.
;       If the "-r exx" option is being given to the command line,
;       then registers bc' de' and hl' need not be saved and restored
;       since the compiler will make no use of them.  The compiler
;       does not use af' in any case.
;*************************************************************************
        region  code
;INTERRUPT
;       push    af              ; save registers
;       push    bc
```

```
;       push    de
;       push    hl
;       push    ix
;       push    iy
;       exx
;       push    bc
;       push    de
;       push    hl
;       exx
;       call    cfcn            ; call some C function
;       exx
;       pop     hl              ; restore registers
;       pop     de
;       pop     bc
;       exx
;       pop     iy
;       pop     ix
;       pop     hl
;       pop     de
;       pop     bc
;       pop     af
;       ei
;       ret                     ; return from interrupt
```

P. FILENAME GLOBAL.C

```
/*  April 19, 1988   global.c  */

#include "solareva.h"

char prtconnected;          /* TRUE is there is a terminal attached,
                               FALSE, otherwise. */

struct datetime clock;      /* The most recently read time will be stored
                               here. */

struct idatetime waketime;  /* The most recently read integer version of
                               time will be stored here. */
```

Q. FILENAME CLOCK.C

```
/*       April 19, 1988          clock.c                  */

#include "solar.h"
#include "convert.h"
#include "inout.h"
#include "solareva.h"
#include "global.h"
#include "newio.h"

void clockint(struct datetime *clock,struct idatetime *iclock);
void clockread(struct datetime *your_clock);
char clockcompare(struct idatetime *clock1,struct idatetime *clock2);
void clockset(struct datetime *clock);
void clocksum(struct idatetime *result,struct idatetime *clock1,
                  struct idatetime *clock2);
void dump_clock(struct datetime *clock);
void rtc(void);
void show_waketime(struct idatetime *waketime);
void testtimeout(void);
char timeout(int delaytime,int measure);


/*************************************************************************/
/* Convert a datetime structure to an idatetime equivalent.  This allows
   arithmetic to be performed on dates and times. */
void clockint(struct datetime *clock,struct idatetime *iclock)
{
         iclock->imonth   = bcd_int(clock->month);
         iclock->idate    = bcd_int(clock->date);
         iclock->ihour    = bcd_int(clock->hour);
         iclock->iminute  = bcd_int(clock->minute);
         iclock->isecond  = bcd_int(clock->second);
}

/*************************************************************************/
/* This routine fills a clock structure with the current date and time. */
void clockread(struct datetime *your_clock)
{
         int     i;

         i = 0;
         do {
                  your_clock->second       = input(SECONDS);
                  your_clock->minute       = input(MINUTES);
                  your_clock->hour = input(HOURS);
                  your_clock->date = input(DATE);
                  your_clock->month        = input(MONTH);
         } while (your_clock->second      != input(SECONDS) && ++i <= 10 * TRIES);
}

/*************************************************************************/
/* Compare two clock times.  Return TRUE if the first is later than the second,
```

```
   FALSE otherwise. */
char clockcompare(struct idatetime *clock1,struct idatetime *clock2)
{
        int     difference;

        difference = clock1->imonth - clock2->imonth;
        /* This logic allows you to decide January comes after December. */
        if ((difference + 12) % 12 < 6
                && difference != 0) return(TRUE);
        if (difference != 0) return(FALSE);
        if (clock1->idate < clock2->idate) return(FALSE);
        if (clock1->ihour < clock2->ihour) return(FALSE);
        if (clock1->iminute < clock2->iminute) return(FALSE);
        if (clock1->isecond < clock2->isecond) return(FALSE);
        return(TRUE);
}

/*************************************************************************/
/* This routine sets the real time clock. */
void clockset(struct datetime *clock)
{
        int     month, date, hour, minute, second, maxdate;
        char    outstr[STRLEN];
        static char     cr[] = "\n\r";

        while (TRUE) {
                portdump("Month? (1-12)  ");
                month = getint();
                if (month >= 1 && month <= 12)
                        break;
                portdump("Invalid month.  Re-enter it.\n\r");
        }
        portdump(cr);
        maxdate = (month == 4    month == 6    month == 9    month == 11) ?
                30 : 31;
        maxdate = (month == 2) ? 28 : maxdate;
        while (TRUE) {
                portdump("Day? (1-");
                itoa(maxdate,outstr);
                portdump(outstr);
                portdump(")  ");
                date = getint();
                if (date >= 1 && date <= maxdate)
                        break;
                portdump("\n rInvalid date.  Re-enter it.\n\r");
        }
        portdump(cr);
        while (TRUE) {
                portdump("Hour? (0-23)  ");
                hour = getint();
                if (hour >= 0 && hour <= 23)
                        break;
                portdump("Invalid hour.  Re-enter it.\n\r");
        }
        portdump(cr);
        while (TRUE) {
```

```
                portdump("Minute? (0-59)  ");
                minute = getint();
                if (minute >= 0 && minute <= 59)
                        break;
                portdump("Invalid minute.  Re-enter it.\n\r");
        }
        portdump(cr);
        while (TRUE) {
                portdump("Second? (0-59)  ");
                second = getint();
                if (second >= 0 && minute <= 59)
                        break;
                portdump("Invalid second.  Re-enter it.\n\r");
        }
        portdump(cr);
        clock->month    = int_bcd(month);
        clock->date             = int_bcd(date);
        clock->hour             = int_bcd(hour);
        clock->minute   = int_bcd(minute);
        clock->second   = int_bcd(second);
        output(MONTH,clock->month);
        output(DATE,clock->date);
        output(HOURS,clock->hour);
        output(MINUTES,clock->minute);
        output(SECONDS,clock->second);
}

/*************************************************************************/
/* Find the sum of two calendar periods. */
void clocksum(struct idatetime *result,struct idatetime *clock1,
                struct idatetime *clock2)
{
        int maxdate;            /* The last valid date in the month. */

        result->isecond = clock1->isecond + clock2->isecond;
        result->iminute = result->isecond / 60;
        result->isecond %= 60;
        result->iminute += clock1->iminute + clock2->iminute;
        result->ihour = result->iminute / 60;
        result->iminute %= 60;
        result->ihour += clock1->ihour + clock2->ihour;
        result->idate = result->ihour / 24;
        result->ihour %= 24;
        result->idate += clock1->idate + clock2->idate;
        result->imonth = 1 + (clock1->imonth + clock2->imonth - 1) % 12;
        maxdate = ((result->imonth == 4)    (result->imonth == 6)
                   (result->imonth == 9)    (result->imonth == 11)) ? 30 : 31;
        /* The real time clock makes no provision for leap year, so leap years
           are ignored in this program (sigh!) */
        maxdate = (result->imonth == 2) ? 28 : maxdate;
        result->imonth += (result->idate - 1) / maxdate;
        result->idate = 1+ (result->idate - 1) % maxdate;
        result->imonth = 1 + (result->imonth - 1) % 12;
}
```

```
/*****************************************************************************/
/* Print a clock structure or dump it to the output port. */
void dump_clock(struct datetime *clock)
{
        if (prtconnected) {
                portdump("Month == ");
                portdump(bcd_asc(clock->month));
                portdump("  Date == ");
                portdump(bcd_asc(clock->date));
                portdump("  Hour == ");
                portdump(bcd_asc(clock->hour));
                portdump("  Minute == ");
                portdump(bcd_asc(clock->minute));
                portdump("  Second == ");
                portdump(bcd_asc(clock->second));
                portdump(".\n\r");
        }
}

/*****************************************************************************/
/* This routine is a menu-driven collection of routines for testing the
   clock functions. */
void rtc(void)
{
        char    data;

        while (TRUE) {
                portdump("\n\rReal time clock functions.\n\r\n\r");
                portdump("A  Read clock.\n\r");
                portdump("B  Set clock.\n\r");
                portdump("C  Test timeout() function.\n\r");
                portdump("Z  Return to main menu.\n\r");

                data = termin();
                echo(data);
                portdump("\n\r");
                switch (data) {
                        case 'a': case 'A':
                                clockread(&clock);
                                dump_clock(&clock);
                                break;
                        case 'b': case 'B':
                                clockset(&clock);
                                break;
                        case 'c': case 'C':
                                testtimeout();
                                break;
                        case 'z': case 'Z':
                                return;
                        default:
                                portdump("Use a valid letter please.\n\r");
                                break;
                }
        }
}
```

```
/*****************************************************************************/
/* This routine displays the wake-up time. */
void show_waketime(struct idatetime *waketime)
{
        char s[STRLEN];  /* String for itoa() routine. */

        portdump("Wake-up time is:  \n\rMonth = ");
        portdump(itoa(waketime->imonth,s));
        portdump("  Date = ");
        portdump(itoa(waketime->idate,s));
        portdump("  Hour = ");
        portdump(itoa(waketime->ihour,s));
        portdump("  Minute = ");
        portdump(itoa(waketime->iminute,s));
        portdump("  Second = ");
        portdump(itoa(waketime->isecond,s));
        portdump("\n\r");
}

/*****************************************************************************/
/* This routine is used to test the timeout() function. */
void testtimeout(void)
{
        char    data,           /* A character entered from the keyboard. */
                units;          /* The units of delay. */
        int     delay;          /* The number of units of delay. */

        while (TRUE) {
                portdump("Test of timeout() function.\n\r\n\r");
                portdump("Specify time units for delay:\n\r\n\r");
                portdump("A  Hours\n\r");
                portdump("B  Minutes\n\r");
                portdump("C  Seconds\n\r");
                portdump("Z  Return to previous menu.\n\r");

                data = termin();
                echo(data);
                switch (data) {
                        case 'a': case 'A':
                                units = HOURS;
                                break;
                        case 'b': case 'B':
                                units = MINUTES;
                                break;
                        case 'c': case 'C':
                                units = SECONDS;
                                break;
                        case 'z': case 'Z':
                                return;
                                break;
                        default:
                                portdump("Use a valid letter please.\n\r");
                                break;
                }
                portdump("\n\rHow many units of delay do you want?\n\r");
                delay = getint();
```

```
                portdump("\n\rStarting delay:\n\r");
                clockread(&clock);
                dump_clock(&clock);
                timeout(delay,units);
                while( !timeout(NULL,NULL) );
                portdump("Delay complete.\n\r");
                echo(BELL);
                clockread(&clock);
                dump_clock(&clock);
        }
}

/*************************************************************************/
/* This routine is used to initiate a timeout sequence, and to test for
   completion.  To set the desired delay time, the parameter "delay"
   should be non-zero.  To test for completion, "delay" should be zero (NULL).
   When setting the delay time, the function always returns TRUE.  When
   testing for completion, it returns TRUE if the time has elapsed, FALSE
   otherwise. */
char timeout(int delaytime,int measure)
        /* "delaytime" is the length of the timeout. */
        /* "measure" is the unit of measure of time.  This can be
                        MONTH, DATE, HOURS, MINUTES, or SECONDS. */
{
        static struct datetime timenow;
        static struct idatetime itimenow, waittime;

        clockread(&timenow);
        clockint(&timenow,&itimenow);
        if (delaytime == NULL) {  /* If delaytime == NULL, then check to
                                        see if timeout period is over. */
                return(clockcompare(&itimenow,&waketime));
        } else {                  /* Otherwise, set the wakeup time. */
                waittime.imonth = waittime.idate = waittime.ihour
                        = waittime.iminute = waittime.isecond = 0;
                switch(measure) {
                        case MONTH:
                                waittime.imonth = delaytime;
                                break;
                        case DATE:
                                waittime.idate = delaytime;
                                break;
                        case HOURS:
                                waittime.ihour = delaytime;
                                break;
                        case MINUTES:
                                waittime.iminute = delaytime;
                                break;
                        case SECONDS:
                                waittime.isecond = delaytime;
                                break;
                }
                clocksum(&waketime,&itimenow,&waittime);
                show_waketime(&waketime);

                return(TRUE);

        }
}
```

R. FILENAME DELAY.S

```
;         May 09, 1988              delay.s

#define   LOOPCOUNT         100

; Delay for n thousands of a second.
; void delay(n)
; int     n;        The number of thousands of seconds of delay desired.

          export    delay
          region    code

delay:    push      ix                ; t=15T.
                                      ; Cause ix to point to the first parameter.
          ld        ix,4              ; t=14T.
          add       ix,sp             ; t=15T.
          ld        c,(ix+0)          ; t=19T.
          ld        b,(ix+1)          ; t=19T.
LOOP1:    ld        de,$LOOPCOUNT     ; t=10T.
LOOP2:    dec       de                ; t= 6T. Count down to zero in LOOP2.
          ld        a,d               ; t= 4T.
          or        e                 ; t= 4T.
          jp        nz,LOOP2          ; t=10T. Inner loop t=24T.
          dec       bc                ; t= 6T. Repeat LOOP1 until time is up.
          ld        a,b               ; t= 4T.
          or        c                 ; t= 4T.
          jp        nz,LOOP1          ; t=10T. Outer loop t=(34+24*LOOPCOUNT)T.
          pop       ix                ; t=14T. Restore ix to its initial value.
          ret                         ; t=10T.
                                      ; Total Delay =(106+(34+24*LOOPCOUNT)*n)T.

;         Solve n ms = (106+(34+24*LOOPCOUNT)*n)T with T = 1/f = 400 ns to
;         get n = LOOPCOUNT. f = 2.5 MHz.  For n=100, LOOPCOUNT = 100, leading
;         to a delay of 97.4 ms for an error of 2.6%.  For n=1,
;         this leads to a delay of 1.016 ms instead of the 1 ms required, for
;         and error of 1.6%.
```

S. FILENAME SYMBOLS

RAMDATA	(specfile)	absol	RAMDATA	818bh	0h
ENDROM	(specfile)	absol	ENDROM	a000h	0h
IRAM	(specfile)	absol	IRAM	a000h	0h
IRAMSZ	(specfile)	absol	IRAMSZ	8h	0h
ENDDATA	(specfile)	absol	ENDDATA	8193h	0h
ZRAM	(specfile)	absol	ZRAM	a008h	0h
ZRAMSZ	(specfile)	absol	ZRAMSZ	106bh	0h
MRAM	(specfile)	absol	MRAM	b073h	0h
MRAMSZ	(specfile)	absol	MRAMSZ	250h	0h
STACKTOP	(specfile)	absol	STACKTOP	10000h	0h
START	code	reloc	START	69h	42h
main	code	reloc	void main();	199h	50h
MBRKPTR	ram	reloc	MBRKPTR	a008h	0h
_fltus	ram	reloc	unsigned char _fltus;	b072h	0h
cellnum	ram	reloc	int cellnum;	a02dh	0h
version	code	reloc	void version();	93h	12h
memory_dump	code	reloc	void memory_dump();	a4h	1bh
testtimeout	code	reloc	void testtimeout();	ce7h	f1h
show_waketime	code	reloc	void show_waketime();	bf1h	deh
itoa	code	reloc	unsigned char *itoa();	1df2h	9ah
atoh	code	reloc	unsigned char atoh();	1965h	16h
ctoh	code	reloc	unsigned char *ctoh();	1cbch	6fh
atoi	code	reloc	int atoi();	1b4ah	3bh
_stod	code	reloc	int _stod();	3655h	dh
printf	code	reloc	int printf();	360fh	1dh
uitoh	code	reloc	unsigned char *uitoh();	1ed6h	b8h
dump	code	reloc	void dump();	ed5h	23h
atohexint	code	reloc	unsigned int atohexint();	1a4eh	2ah
i	ram	reloc	int i;	a035h	0h
waketime	ram	reloc	struct waketime;	b064h	0h
clockcompare	code	reloc	unsigned char clockcompare();	315h	36h
clocksum	code	reloc	void clocksum();	735h	8bh
clockint	code	reloc	void clockint();	1adh	1ah
clockset	code	reloc	void clockset();	42fh	48h
c_bits	ram	reloc	struct *c_bits;	a044h	0h
_vsgn	code	reloc	_vsgn	7a28h	bh
timeout	code	reloc	unsigned char timeout();	dfch	127h
dump_clock	code	reloc	void dump_clock();	a77h	a6h
_ultos	code	reloc	_ultos	36f7h	29h
output	code	reloc	int output();	1952h	12h
currentout	ram	reloc	float currentout;	a03eh	0h
clockread	code	reloc	void clockread();	264h	25h
clock	ram	reloc	struct clock;	b05dh	0h
rtc	code	reloc	void rtc();	b3dh	bah
echo	code	reloc	void echo();	1082h	4ch
inithardware	code	reloc	void inithardware();	1434h	dh
_muld	code	reloc	int _muld();	47e3h	9ch
currentdata	ram	reloc	unsigned char currentdata;	a038h	0h
oldcurrent	ram	reloc	unsigned char oldcurrent;	a043h	0h
voltdata	ram	reloc	unsigned char voltdata;	a037h	0h
portdump	code	reloc	void portdump();	131eh	97h
input	code	reloc	unsigned char input();	1942h	7h
retrieve	code	reloc	void retrieve();	16eeh	71h
cell	ram	reloc	int cell;	a031h	0h

```
menu             code    reloc   unsigned char menu();  f4h        2ah
_mulww           code    reloc   _mulww                 993h       bh
_dtos            code    reloc   int _dtos();           369ch      17h
oldvolt          ram     reloc   unsigned char oldvolt;a042h       0h
cnvgncdone       ram     reloc   unsigned char cnvgncdone;a039h    0h
gethexint        code    reloc   unsigned int gethexint();1187h   6eh
testinput        code    reloc   void testinput();    138fh      afh
delay            code    reloc   void delay();        1f6fh      ch
_modsww          code    reloc   _modsww              4a7ch      14h
experiment_data  ram     reloc   struct experiment_data¢100h|¢8h|;a046h 0h
execute          code    reloc   void execute();      152ch      2eh
bcd_asc          code    reloc   unsigned char *bcd_asc();1be0h  4ch
bcd_int          code    reloc   int bcd_int();       1c49h      60h
prtconnected     ram     reloc   unsigned char prtconnected;b05ch 0h
int_bcd          code    reloc   unsigned char int_bcd();1d64h   89h
termin           code    reloc   unsigned char termin();1373h   a4h
getint           code    reloc   int getint();    126ch        84h
testoutput       code    reloc   void testoutput();13e9h       c0h
gethex           code    reloc   unsigned char gethex();10a4h   57h
checkprt         code    reloc   unsigned char checkprt();ed2h  17h
_divwws          code    reloc   _divwws              1fach      16h
command          data    reloc   struct command;      a003h      0h
voltage          ram     reloc   int voltage;         a02fh      0h
tolower          code    reloc   unsigned char tolower();1eaah  aeh
voltout          ram     reloc   float voltout;       a03ah      0h
row              ram     reloc   int row;            a033h      0h
_subd            code    reloc   int _subd();        4509h      6ch
_stol            code    reloc   int _stol();        3ee5h      23h
_round           data    reloc   int _round;         a004h      0h
_tstd            code    reloc   int _tstd();        3fe0h      12h
_sltoa           code    reloc   unsigned char *_sltoa();5577h  14h
_shrul           code    reloc   int _shrul();       6306h      1ch
_shlul           code    reloc   int _shlul();       63f2h      2ah
_sltos           code    reloc   _sltos              36d6h      28h
_tstmd           code    reloc   int _tstmd();       4036h      19h
strlen           code    reloc   int strlen();       5543h      9h
strchr           code    reloc   unsigned char *strchr();5507h  ch
_stosgl          code    reloc   int _stosgl();      7329h      dh
_shrull          code    reloc   int _shrull();      57eah      16h
_shlull          code    reloc   int _shlull();      59eah      2ah
_sgltos          code    reloc   int _sgltos();      750fh      21h
_addd            code    reloc   int _addd();        4106h      30h
_ultoa           code    reloc   unsigned char *_ultoa();56e3h  14h
_ltod            code    reloc   int _ltod();        3c9dh      eh
_ltos            code    reloc   int _ltos();        3e45h      eh
_addul           code    reloc   int _addul();       64d5h      37h
_dblprec         data    reloc   int _dblprec;       a006h      0h
_uprint          code    reloc   int _uprint();      2048h      34h
_cmpd            code    reloc   int _cmpd();        454bh      79h
_cmpl            code    reloc   _cmpl               6b35h      dh
_negd            code    reloc   int _negd();        4095h      27h
_negl            code    reloc   _negl               7a11h      ah
_addull          code    reloc   int _addull();      5be1h      3eh
_mulul           code    reloc   int _mulul();       65cfh      4bh
_mulll           code    reloc   _mulll              796dh      18h
_divd            code    reloc   int _divd();        48d6h      aeh
```

_dtol	code	reloc	int _dtol();	3d4ah	24h
uctype	const	reloc	unsigned const char uctype¢81h¦;7b54h		0h
_normd	code	reloc	int _normd();	4aa4h	eh
_nrmul	code	reloc	int _nrmul();	6568h	41h
_norms	code	reloc	int _norms();	5181h	eh
_ecvt	code	reloc	unsigned char *_ecvt();3bech		2bh
_mulull	code	reloc	int _mulull();	5e1ch	5ah
_modsll	code	reloc	_modsll	78e2h	29h
_modull	code	reloc	_modull	7882h	ch
_mulwul	code	reloc	_mulwul	79a4h	13h
_mulwsl	code	reloc	_mulwsl	79c1h	1bh
_moduww	code	reloc	_moduww	4a61h	ch
_dtosl	code	reloc	_dtosl	3c66h	24h
_dtoul	code	reloc	_dtoul	3c82h	25h
_divul	code	reloc	int _divul();	6a21h	5dh
fputc	code	reloc	int fputc();	7a38h	10h
_fcvt	code	reloc	unsigned char *_fcvt();3c29h		35h
_nrmull	code	reloc	int _nrmull();	5d48h	49h
_zerod	code	reloc	int _zerod();	405ch	1fh
_dbltoa	code	reloc	unsigned char *_dbltoa();3718h		16h
_dbltod	code	reloc	int _dbltod();	6e1ch	22h
_dtodbl	code	reloc	int _dtodbl();	6b52h	dh
_divull	code	reloc	int _divull();	6053h	71h
_divlls	code	reloc	_divlls	77f8h	2ch
_divwwu	code	reloc	_divwwu	1f8fh	ch
_divllu	code	reloc	_divllu	7795h	ch

APPENDIX C. SOLAR CELL ARRAY TEST CIRCUIT CODE

A. FILENAME CELLTEST.C

```
/*************************************************************************/
/*    May 28, 1988   Celltest.c
Program for testing solar cell array I-V characteristics  */



#include "solareva.h"
#include "newio.h"
#include "convert.h"
#include "delay.h"

#define ARRAYSZ  1          /* number of test cell */
#define STOP     0          /* stop */
#define START    3          /* high assertion for two bits (convergence) */

int cellnum, voltage, cell, row, i;
char voltdata;
char currentdata;
char cnvgncdone;
float voltout;
float currentout;
char oldvolt;
char oldcurrent;

void execute(void);
void retrieve(void);

struct PORT1_B {                        /*PortB-1 Bit field */
              unsigned int un:      3;  /*unused bit*/
              unsigned int strtcn:  2;  /*a/d start conversion signal*/
              unsigned int celladd: 3;  /*solar cell number*/
} command = {0,0,0};               /* initialize PortB_1 */

struct C_PORT {                         /*Bit field for Port C-2 input */
             unsigned int hi:   5;      /*unused high bits*/
             unsigned int bits: 2;      /*bits C1 and C2*/
             unsigned int lo:   1;      /*unused low bits*/
} *c_bits;

struct data_pt {                       /*Array structure for data storage*/
             char voltagept;           /*voltage*/
             char currentpt;           /*current*/
} experiment_data[256] [8];            /*row/column for data*/

void execute(void)
{
```

```
for (cellnum=0; cellnum < ARRAYSZ; cellnum++){    /*loop for test cell*/

  row=0;                                          /*storage counter*/
  oldvolt=0;                                      /*initialize comparison variable*/
  oldcurrent=0;                                   /*initialize comparison variable*/

  command.celladd=cellnum;                        /*set address bits*/
  output(PORT1_CTRL, command);                    /*initialize PortB-1*/

  for (voltage=0; voltage < 256; voltage++){      /*input voltage ladder*/
     output(PORT1_DA, voltage);                   /*output cell bias voltage*/

     command.strtcn=START;                        /*Start Convergence pulse*/
                                                       /*for both ADC's*/
     output(PORT1_CTRL, command);                 /*do it*/

     command.strtcn=STOP;                         /*End Start Convergence*/
                                                  /*pulse for both ADC's*/
     output(PORT1_CTRL, command);                 /*do it*/

     delay(1);                                    /*1 ms delay for settling*/

     while (TRUE){                                /*EOC check loop*/

       cnvgncdone = input(PORTC2);                /*assign PORTC2 word*/
       c_bits = (struct C_PORT *)(&cnvgncdone);   /*looking for EOC bits*/
         if (c_bits->bits==0x03)                   /*bits C1 and C2 must be high*/
          break;                                  /*When EOC bits high, cont.*/
     }

     voltdata = input(PORT2_ADV);                 /*collect voltage info*/
     currentdata = input(PORT2_ADC);              /*collect current info*/

     if((voltdata == oldvolt) && (currentdata == oldcurrent))
       continue;                                  /*ignore transistor bias*/
                                                  /*and multiple data*/

     if(oldvolt != 0){                            /*smooth curve, delete*/
      if(oldvolt < voltdata)                      /*voltage surges*/
        continue;
     }

     oldvolt=voltdata;                            /*reset comparison*/
     oldcurrent=currentdata;                      /*reset comparison*/

     if(voltdata == 0)                            /*Isc reached*/
       voltage = 255;                             /*end loop*/

 /* Data Storage */

     experiment_data[row][cellnum].voltagept = voltdata;
                                                       /*stores voltage data*/
     experiment_data[row][cellnum].currentpt = currentdata;
                                                       /*stores current data*/
     row++;                                            /*increment array row*/
```

```
    }
  }
  voltage=0;                                        /*turn off bias*/
  output(PORT1_DA, voltage);                        /*do it*/
}
/*****************************************************************************/
/* routine to retreive data from RAM */

void retrieve(void)
{
  printf("Specify cell number; 0-7.  ");    /*Which cell data?*/

  cell = getint();                          /*get cell number from terminal*/

  printf("\n\r");

  printf("%d,%d\n\r",2,2);                  /*disc file output header*/

  for (i=0; i<256 && ((experiment_data[i][cell].voltagept != 0)
                    (experiment_data[i][cell].currentpt != 0)); ++i){
                                    /*loop prevents collecting data past end of file*/

      voltout = (float)experiment_data[i][cell].voltagept * .0041;
                                    /*floating decimal at .0041 mv per step*/
      currentout = (float)experiment_data[i][cell].currentpt * .00117;
                                    /*floating decimal at .0117 mv per step and*/
                                    /*division by 9.9 ohms to current*/

      printf("%f,", voltout);           /*output voltage*/
      printf("%f,", currentout);        /*output current*/
      printf("%f\n\r", 0.0);           /*disk file trailer*/
  }
  printf("%d,%d,%d",30,30,30);          /*disk file end parameters*/
}
```

APPENDIX D. SAMPLE SILICON SOLAR CELL TEST DATA

A. FILENAME SILICON.DAT

```
2,2
0.524800,0.000000,0.000000
0.524800,0.001170,0.000000
0.524800,0.002340,0.000000
0.524800,0.003510,0.000000
0.524800,0.004680,0.000000
0.524800,0.005850,0.000000
0.524800,0.007020,0.000000
0.524800,0.008190,0.000000
0.524800,0.009360,0.000000
0.524800,0.010530,0.000000
0.524800,0.011700,0.000000
0.524800,0.012870,0.000000
0.524800,0.014040,0.000000
0.524800,0.015210,0.000000
0.524800,0.016380,0.000000
0.524800,0.017550,0.000000
0.524800,0.018720,0.000000
0.524800,0.019890,0.000000
0.524800,0.021060,0.000000
0.524800,0.022230,0.000000
0.524800,0.023400,0.000000
0.524800,0.024570,0.000000
0.524800,0.025740,0.000000
0.524800,0.026910,0.000000
0.524800,0.028080,0.000000
0.524800,0.029250,0.000000
0.524800,0.030420,0.000000
0.524800,0.031590,0.000000
0.524800,0.032760,0.000000
0.520700,0.033930,0.000000
0.520700,0.035100,0.000000
0.520700,0.036270,0.000000
0.516600,0.037440,0.000000
0.516600,0.038610,0.000000
0.512500,0.039780,0.000000
0.512500,0.040950,0.000000
0.512500,0.042120,0.000000
0.508400,0.043290,0.000000
0.508400,0.044460,0.000000
0.508400,0.045630,0.000000
0.504300,0.046800,0.000000
0.504300,0.047970,0.000000
0.504300,0.049140,0.000000
0.500200,0.050310,0.000000
0.500200,0.051480,0.000000
0.500200,0.052650,0.000000
0.496100,0.053820,0.000000
```

0.496100,0.054990,0.000000
0.492000,0.056160,0.000000
0.492000,0.057330,0.000000
0.492000,0.058500,0.000000
0.487900,0.059670,0.000000
0.487900,0.060840,0.000000
0.487900,0.062010,0.000000
0.483800,0.063180,0.000000
0.483800,0.064350,0.000000
0.483800,0.065520,0.000000
0.479700,0.066690,0.000000
0.479700,0.067860,0.000000
0.475600,0.069030,0.000000
0.475600,0.070200,0.000000
0.475600,0.071370,0.000000
0.471500,0.072540,0.000000
0.471500,0.073710,0.000000
0.467400,0.074880,0.000000
0.467400,0.076050,0.000000
0.467400,0.077220,0.000000
0.463300,0.078390,0.000000
0.463300,0.079560,0.000000
0.463300,0.080730,0.000000
0.455100,0.081900,0.000000
0.455100,0.084240,0.000000
0.451000,0.085410,0.000000
0.451000,0.086580,0.000000
0.451000,0.087750,0.000000
0.446900,0.088920,0.000000
0.446900,0.090090,0.000000
0.442800,0.091260,0.000000
0.438700,0.092430,0.000000
0.438700,0.094770,0.000000
0.434600,0.095940,0.000000
0.430500,0.098280,0.000000
0.430500,0.099450,0.000000
0.430500,0.100620,0.000000
0.422300,0.101790,0.000000
0.422300,0.102960,0.000000
0.422300,0.104130,0.000000
0.414100,0.105300,0.000000
0.414100,0.107640,0.000000
0.405900,0.108810,0.000000
0.397700,0.112320,0.000000
0.385400,0.115830,0.000000
0.377200,0.119340,0.000000
0.360800,0.122850,0.000000
0.336200,0.126360,0.000000
0.295200,0.129870,0.000000
0.028700,0.131040,0.000000
0.000000,0.132210,0.000000
30,30,30

LIST OF REFERENCES

1. Green, Martin A., *Solar Cells: Operating Principles, Technology, and System Applications.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

2. Rauschenbach, H. S., *Solar Cell Array Design Handbook, Volume 1.* Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Ca., Publication SP 43-38, Vol.1, October 1976.

3. Anspaugh, Bruce, *A Verified Technique for Calibrating Space Solar Cells.* Conference Record of the 19th IEEE Photovoltaic Specialists Conference, 1987. IEEE, New York, N.Y., 1987, 542-547.

4. Tada, H. Y., J. R. Carter, Jr., B. E. Anspaugh, and R. G. Downing, *Solar Cell radiation Handbook.* Jet Propulsion Laboratory, Pasedena, Ca., Publication 82-69, November 1982.

5. Private communication with Lt. S. Sage, Naval Postgraduate School, Monterey, Ca., April 1988.

6. Statler, R. L., and D. H. Walker, *NTS-2 Solar Cell Experiment After Two Year in Orbit,* The Conference Record of the Fourteenth Photovoltaic Speicalists Conference, 1980. IEEE, New York, N.Y., 1980, pp.1234-1239.

7. Trumble, Terry M., and Fred Betz, *Evaluation of a Gallium Arsenide Solar Panel on the LIPS II Satellite,* The Conference Record of the Seventeenth Photovoltaic Specialists Conference, 1984, IEEE, New York, N.Y., May 1984, pp.1108-1111.

8. Gussenhoven, M.S., Ed., *CRRES/SPACERAD Experiment Descriptions,* Space Physics Division, Air Force Geophysics Laboratory, Hanscom AFB, Ma. September, 1984.

9. Morris, Lt. Robert K., *New Insight into the LIPS-II GaAs Solar Panel Performance,* The Conference Record of the Eighteenth Photovoltaic Specialists Conference, 1985. IEEE, New York, N.Y., 1985, 688-691.

10. Suppa, E. G., *Space Calibration of Solar Cells. Results of 2 Shuttle Flight Missions,* The Conference Record of the Seventeenth Photovoltaic Specialists Conference, 1984. IEEE, New York, N.Y., 1984, 301-305.

11. Callaway, Robert K., *An Autonomous Circuit for the Measurement of Photovoltaile Devices Parameters,* M.S. Thesis, Naval Postgraduate School, Monterey, Ca., 1986.

12. National Semiconductor Corporation, *NSC800 High-Performance Low-Power Microprocessor,* Santa Clara, Ca., 1983.

13. Wallin, Jay W., *Microprocessor Controller With Nonvolatile Memory Implementation,* M.S. Thesis, Naval Postgraduate School, Monterey, Ca., 1985.

14. National Semiconductor Corporation, *NSC810A RAM-I/O-Timer,* Santa Clara, Ca., 1984.

15. National Semiconductor Corporation, *MM58167 Microprocessor Compatible Real Time Clock,* Santa Clara, Ca., 1984.

16. INTEL Corporation, *27C64/87C64 64K (8Kx8) CHMOS Production and UV Erasable PROMS,* Santa Clara, Ca., June 1986.

17. Radio Company of America (RCA), *CDM6264 CMOS 8192-Word by 8-Bit LSI Static RAM,* Somerville, N.J., January 1985.

18. National Semiconductor Corporation, *DAC0800 8-bit Digital-to-Analog Converter,* Santa Clara, Ca., 1982.

19. Harris Semiconductor Corporation, *HI-506A Single 16/Differential 8 Channel CMOS Analog Multiplexer,* Melbourne, Fl., 1985.

20. National Semiconductor Corporation, *ADC0809 8-Bit Microprocessor Compatible A/D Converter with 8-Channel Multiplexer,* Santa Clara, Ca., 1982.

21. Cameron, Charles B., *Control of an Experiment to Measure Acoustic Noise in the Space Shuttle,* M.S. Thesis, Naval Postgraduate School, Monterey, Ca., 1988.

22. Gold, Don W., *High Energy Electron Radiation Degradation of Gallium Arsenide Solar Cells,* M.S. Thesis, Naval Postgraduate School, Monterey, Ca., 1986.

23. Young, Thomas, *Linear Systems and Digital Signal Processing,* Prentice-Hall, Englewood Cliffs, N.J., 1985.

BIBLIOGRAPHY

Hu, Chenming, and Richard M. White, *Solar Cells, From Basics to Advanced Systems,* McGraw-Hill Book Co., New York, N.Y., 1983.

Jet Propulsion Laboratory, *Solar Cell Array Design Handbook, Vol.2,* NASA, Jet Propulsion Laboratory, Pasadena, Ca., Publication SP 43-38 Vol.2, October 1976.

Purdam, Jack, *C Programming Guide,* Que Corporation, Indianapolis, In., 1985.

Sedra, Adel S., and Kenneth C. Smith, *Microelectronic Circuits,* The Dryden Press, Saunders College Publishing, New York, N.Y., 1982.

Stone, Harold S., *Microcomputer Interfacing,* Addison-Wesley Publishing Co., Reading, Mass., 1982.

Taub, Herbert, *Digital Circuits and Microprocessors,* McGraw-Hill Book Co., New York, N.Y., 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor S. Michael Code 62Mi Naval Postgraduate School Monterey, CA 93943	2
4.	Professor R. Panholzer Code 62Pz Naval Postgraduate School Monterey, CA 93943	2
5.	Professor O. Heinz Code 61Hz Naval Postgraduate School Monterey, CA 93943	1
6.	Professor K. Kartchner Code 56Kn Naval Postgraduate School Monterey, CA 93943	1
7.	Superintendent Code 39 Naval Postgraduate School Monterey, CA 93943	1
8.	Professor J. Powers Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943	2
9.	Department of the Navy Commander Space and Naval Warfare Systems Command, PDW 106-483 Attn: Lcdr. R. Harding Washington, D.C. 20363-5100	2

10. Lt. Robert R. Oxborrow — 4
Tactical Electronic Warfare Squadron 129
Naval Air Station Whidbey Island
Oak Harbor, WA 98278-6100

11. Navy Space System Division — 1
Chief of Naval Operations (OP-943)
Washington, D.C. 20305-2000

12. Commander, United States Space Command — 1
Attn: Technical Library
Peterson AFB, CO 80914

13. Commander, Naval Space Command — 1
Attn: Code N3
Dahlgren, VA 22448

14. National Aeronautics and Space Administration — 1
Technical Library
NASA Headquarters
600 Independence Ave.
Washington, D.C. 20546

15. Naval Research Laboratory — 1
Condensed Matter and Radiation Science Division
(Code 4612)
Mr. Richard L. Statler
Washington, D.C. 20375-5000

Thesis
0982
c.1

Oxborrow
A microprocessor-based,
solar cell parameter
measurement system.